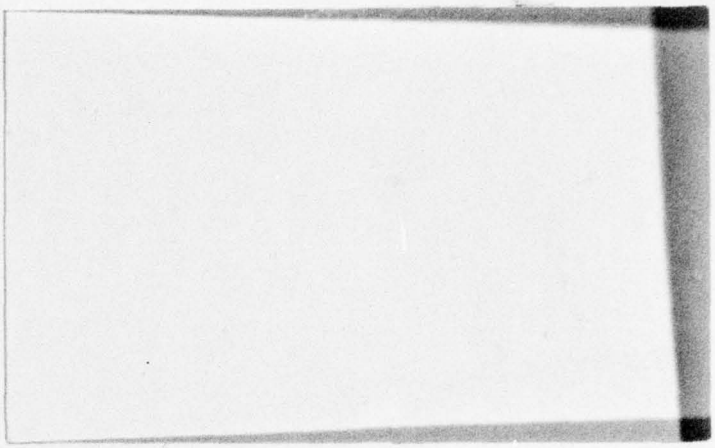
AD-A065 096 TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES F/G 15/5
IMPLEMENTATION AND ANALYSIS OF A VARIANT OF THE DUAL METHOD FOR--ETC(U)
OCT 78 R D ARMSTRONG, D KLINGMAN, D WHITMAN N00014-78-C-0222
UNCLASSIFIED CCS-324 NL

| OF |
ADA
065096

END
DATE
FILMED
4-79
DDC

LEVEL II

12 B.S.





# CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

(7)

(14) Research Report CCS-324

(6) IMPLEMENTATION AND ANALYSIS OF A VARIANT OF THE DUAL METHOD FOR THE CAPACITATED TRANSSHIPMENT PROBLEM.

by

(10) Ronald D. Armstrong*
Darwin Klingman**
David Whitman***

(9) Research rept.,

(11) October 1978 (12) 66p.

*Professor of Statistics/Operations Research, University of Texas, BEB 600, Austin, Texas, 78712.

**Professor of Operations Research and Computer Sciences and Director of Computer Science Research, Center for Cybernetic Studies, University of Texas, BEB 608, Austin, Texas 78712.

***Research Assistant, Department of General Business, University of Texas, BEB 600, Austin, Texas 78712

(15) This research was partly supported by Project No. NR047172, ONR Contract N00014-78-C-0222, and NSF Contract MCS77-00100 with the Center for Cybernetic Studies, The University of Texas. 

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 203E
The University of Texas
Austin, Texas 78712
(512) 471-1821

406 197

LB

## ABSTRACT

This paper presents a variant of the dual simplex method for the capacitated pure network problem and a computational analysis of this algorithm. This work includes the considerations of different list structures to store the original problem data and the basis and the testing of various procedures to select the leaving basic variable. This study also examines the sensitivity of the implementation to changes in problem parameters. The results show that the algorithm which is presented here is superior to earlier dual implementations.

Implementation and Analysis of a Variant of the
Dual Method for the Capacitated Transshipment Problem

This paper describes the development of a variant of the dual simplex algorithm for the capacitated pure network problem and presents the results of an extensive computational analysis of this algorithm. Comparisons of the dual method with primal and out-of-kilter codes in the early 1970's indicated that the dual method was slower by approximately an order of magnitude in solving capacitated network problems [3, 12]. Therefore, most of the network research over the last ten years has been focused only on special purpose primal simplex and out-of-kilter algorithms and codes. We were able to locate only three dual transshipment codes, DNET [12], DNET-I [15], and [16].

For highly capacitated transshipment problems, the implementation of the variant of the dual simplex method which is presented here is superior to the earlier dual codes. However, even for these problems, the special purpose primal simplex method is superior both in terms of solution time and central memory requirements.

Three aspects of the dual simplex algorithm were examined and modified in order to improve its efficiency. First, a number of list structures for storing the original problem data and the basis (Section 4) were considered. Next, we sought to implement, in the capacitated pure transshipment setting, a modification of the dual method which incorporates a multiple pivot strategy (Section 2.1). Finally, different procedures for the selection of the arc to leave the basis (Section 5.2)

were tested.

The list structures tested and evaluated for the new dual code were chosen with two objectives in mind. They had to facilitate efficient performance of the steps of the algorithm and they had to obviate the processing of non-relevant data. We first tested several of the recently developed, highly efficient linked list structures which optimize the implementation of the primal simplex network algorithm. These sophisticated structures were not available when DNET was originally developed.

We tested other list structures which would allow one to identify and thus avoid nonbasic variables which are ineligible to enter the basis during an iteration of the dual simplex method. This process is equivalent to performing a partial rather than a complete LP tableau row scan to determine the incoming variable for each dual pivot in the general linear programming setting.

Since even a partial scan for each dual pivot is quite time consuming, we modified the dual method by incorporating a particular multiple pivot strategy for each partial scan. This strategy generalizes a technique used by Barrodale and Roberts to solve $L_1$ norm problems [4]. (Multiple pivot approaches have also been considered by Balas [1] and Witzgall [20] to solve linear programs).

The final aspect of this work was to analyze the sensitivity of the resultant dual code to variations in problem parameter values and to compare its performance to that of a state-of-the-art primal capacitated pure transshipment code over a set of problems especially well suited to the dual code. The results of all our computational testing are reported and explained in Section 6 of this paper.

## 1.1 PROBLEM STATEMENT

The *capcitated transshipment (CT) problem*, may be defined as follows:

Primal

$$\text{Minimize} \quad c^T x \tag{1}$$

subject to

$$Ax = b \tag{2}$$

$$0 \leq x \leq u \tag{3}$$

Dual

$$\text{Maximize} \quad w^T b - v^T u \tag{4}$$

subject to

$$w^T A - v \leq c^T \tag{5}$$

$$w \text{ unrestricted} \tag{6}$$

$$v \geq 0$$

where u is an nx1 positive vector and A is an mxn matrix whose columns each contain exactly two nonzero coefficients, -1 and 1. Note that the primal problem is equivalent, by a simple translation of variables, to the seemingly more general problem in which the lower bound on x is different from zero.

## 2.0 THE DUAL METHOD

In this section, the steps of the dual method in a general (bounded variable) minimization linear programming setting are presented [11, 19]. In Section 2.1, the modification of the dual method to incorporate a multiple pivot strategy is indicated. The remainder of the

paper then investigates the specialization of the multiple pivot strategy to solve CT problems.

Lemke's dual method may be succinctly described as follows:

1. Begin with a dual feasible basis B and the vector of cost coefficients $c_B$. Determine the initial primal solution, $x_B$.
2. Select a basic variable (call it $x_\ell$) whose value violates either its upper bound or lower bound of zero. If no such variable exists, the current basic solution is optimal and the procedure terminates.
3. Determine the updated linear equation expressing $x_\ell$ as a linear combination of the current nonbasic variables; i.e., $x_\ell = \lambda_o + \sum_{k\epsilon NB} \lambda_k(-x_k)$, where NB denotes the index set for the current nonbasic variables.
4. Let $\lambda_k^1 = -\lambda_k$ for $k\epsilon NB$ if $x_k$ is set equal to its lower bound and $x_\ell$ violates its lower bound, or if $x_k$ is set equal to its upper bound and $x_\ell$ violates its upper bound. For the remaining $k\epsilon NB$, let $\lambda_k^1 = \lambda_k$, and form $NB^+ = \{k\epsilon NB \mid \lambda_k^1 > 0\}$. If $NB^+ = \phi$, then the problem is primal infeasible and the procedure terminates.
5. Express the updated equation of the objective function variable to be minimized (call it $x_o$) in terms of the current nonbasic variables; i.e., $x_o = \pi_o + \sum_{k\epsilon NB} \pi_k(-x_k)$, where $\pi_k = c_B B^{-1} A_k - c_k = wA_k - c_k$, and $A_k$ denotes the kth column of the original coefficient matrix A.

6. Identify a nonbasic variable $x_s$, $s \in NB^+$, such that $|\pi_s/\lambda_s| = \min_{k \in NB^+} \{|\pi_k/\lambda_k|\}$.

7. Determine a new current dual feasible basic solution by removing s and adding $\ell$ to NB (making $x_s$ basic and $x_\ell$ nonbasic), and assigning $x_\ell$ the value of the bound it previously violated while holding the other nonbasic variables constant (identifying the values thus assigned to the new set of current basic variables). Then return to Step 2.

The linear equation of Step 3 for the leaving variable may be obtained by using the poly-$\omega$ technique of Charnes and Cooper [5]. According to this technique, the coefficients of the equation defining $x_\ell$ in Step 3 correspond to ${}_pA^*$, the pth row of the updated coefficient matrix A*, where $A^* = B^{-1} A$. One may generate ${}_pA^*$ by first premultiplying $B^{-1}$ by a vector $\tilde{c}_B$ where $\tilde{c}_\ell = 1$ and $\tilde{c}_k = 0$, for $k \neq \ell$; one then post multiplies the resulting vector by the original coefficient matrix A. $\tilde{c}_B B^{-1}$ will be referred to as the vector of "pseudo-dual" variable values and denoted $\tilde{w}$. The $\lambda_k$'s of Step 3 may be defined as $\lambda_k = \tilde{w}A_k - \tilde{c}_k$. Hence, the $\pi_k$'s and $\lambda_k$'s are expressable in a similar form.

Since most linear programs have many more columns than rows, |NB|, the cardinality of NB, is rather large. Therefore, the selection of the entering variable at each iteration is a time consuming process. Further, it is an inherently wasteful process since at each iteration $NB^+$ must be formed in order to select the entering variable. $NB^+$ is then discarded.

## 2.1 MULTIPLE PIVOT STRATEGY

Several researchers have tried to improve the efficiency of the bounded variable dual method by utilizing a "multiple pivot" strategy (Balas [1] and Witzgall [20]). Specifically, if $x_s$, the variable selected to enter the basis, will do so at a primal infeasible level, then one should not pivot $x_s$ into the basis. Rather the value of $x_s$ should be changed from its current bound to the other bound. Furthermore, $I_\ell$, which denotes the *infeasibility* of the variable $x_\ell$ which has been selected to leave the basis, should be decreased by $|\lambda_s u_s|$. The values of the other variables are not changed. Next, one must delete s from $NB^+$ and repeat Step 6. One continues in this fashion until $NB^+ = \phi$ or until the variable selected to enter the basis, $x_{s_r}$ (where $r > 1$), will do so at a feasible level. If $NB^+ = \phi$, then the problem is primal infeasible. On the other hand, if $x_{s_r}$ will enter the basis at a primal feasible level, then one should permit it to do so and perform Step 7. The following theorem shows that the resulting solution is dual feasible.

Theorem 1: Let the indices $s \equiv s_1, s_2, \ldots, s_r$ be determined as previously described. The sequence of operations defined by the modified dual algorithm will provide a dual feasible basis after the completion of Step 7.

Proof. It follows from the formula for updating the $\pi_k$'s that (1) the sign of $\pi_k$ will not change for $k \notin NB^+$ or for $k \in NB^+$ with $|\pi_k/\lambda_k| > |\pi_{s_r}/\lambda_{s_r}|$, (2) the sign of $\pi_k$ will change for $k \in NB^+$ with $|\pi_k/\lambda_k| < |\pi_{s_r}/\lambda_{s_r}|$, and (3) the $\pi_k$ will become zero for $k \in NB^+$ with $|\pi_k/\lambda_k| = |\pi_{s_r}/\lambda_{s_r}|$.

The only nonbasic variables to change values during an iteration are $x_{s_1}, x_{s_2}, \ldots, x_{s_{r-1}}$ and they will switch in value from one bound to the other. However, by construction

$$|\pi_{s_i}/\lambda_{s_i}| \leq |\pi_{s_r}/\lambda_{s_r}|, \; i = 1, \ldots, r-1.$$

Therefore, the sign of $\pi_{s_i}$ changes, or $\pi_{s_i}$ goes to zero, for $i = 1, \ldots, r-1$, during the update. The sign must change in order to preserve dual feasibility. Since all other nonbasic $x_k$ satisfy $k \notin NB^+$ or $|\pi_k/\lambda_k| \geq |\pi_{s_r}/\lambda_{s_r}|$, the solution will be dual feasible after the completion of step 7.

Note that $NB^+$ need be computed only once during an iteration. $NB^+$ may be sorted to identify $s_1, \ldots, s_r$. Then $x_{s_1}, \ldots, x_{s_{r-1}}$ are set to their other bound, and $x_{s_r}$ enters the basis. Finally, if $r = 1$, the multiple pivot is merely a single pivot; in this case the modified dual algorithm coincides with the standard dual method.

## 3.0 GRAPHICAL APPROACH

The most efficient procedures for solving capacitated transshipment problems [10, 12, 14] are based on viewing the problem in a graphical context. The modified dual algorithm is integrated with these procedures, which are adaptations of the simplex algorithm [5, 6, 10] in which the A matrix and the basis matrix are stored as graphs using computer list structures. The use of such structures reduces both the amount of work needed to perform the simplex operations and the amount of computer memory required to store essential problem data. In addition, the graphs

contain only the nonzero matrix components which allow special graph labeling procedures [7, 10, 13]. This eliminates checking and unnecessary arithmetic operations on zero elements. Further, the graphical representation allows one to characterize fully the nonzero elements of the representation of any nonbasic vector and the signs of these elements. Finally, this representation allows one to characterize explicitly which dual values change and how these values are altered after performing a basis exchange step.

The essential concepts and definitions of elementary graph theory which are used in our development are outlined and related to the capacitated transshipment problem in the next section.

## 3.1 GRAPHICAL INTERPRETATION OF THE CT PROBLEM

A directed graph G(N,E) is a finite set of *nodes* N and *arcs* E connecting the nodes. Each arc is represented by an ordered pair (i,j), which defines the arc from node i to node j. It is not necessary for all pairs of nodes to be joined and there may be multiple arcs between the same two nodes.

The capacitated transshipment problem defines a directed graph as follows. Each row of A corresponds to a node and each column corresponds to an arc of the graph. Associated with each arc is a *variable*, an *upper bound* and an *objective function* coefficient. The value of the variable (i.e., the component of x) associated with the arc is called the *flow* on the arc. The row positions of the nonzero entries (+1 and -1) in the column which corresponds to the arc identify

the nodes on which the arc is incident. It will be assumed, henceforth, that the arc is directed from the node associated with the -1 term to the node associated with the +1 term. Additionally, an arc directed from node i to node j will be denoted arc (i,j) and $x_{ij}$, $c_{ij}$, and $u_{ij}$ denote arc (i,j)'s flow value, cost, and upper bound, respectively.

Since the index ij naturally refers to arc (i,j), double subscripting will be henceforth employed to improve readibility of the subsequent discussion. In particular $A_{ij}$, $\lambda_{ij}$, $\lambda^{1}_{ij}$, $\pi_{ij}$ and $I_{ij}$ will be employed rather than $A_k$, $\lambda_k$, $\lambda^{1}_k$, $\pi_k$, and $I_k$. Further, the index ij rather than k will denote an element of NB or $NB^+$. Technically a third index should be used, since there may be more than one arc from i to j. However, for notational convenience, the third subscript will be omitted. The method subsequently described provides an organization by which multiple arcs with unique costs and upper bounds are readily accommodated.

The right hand side vector b for the capacitated transshipment problem associates a node requirement $b_k$ (the kth component of b) with node k of the graph. (Each unit of flow on an arc (i,j) therefore "contributes" -1 and 1, respectively, to the node requirements $b_i$ and $b_j$.) Note that a negative (positive) node requirement then corresponds to a supply (demand) requirement at the node.

The w vector of Step 5 (Section 2.0) is referred to as a vector of dual variables. Given a current basis B, the values of the individual components of w which satisfy complementary slackness may be determined algebraically in the following manner.

For each basic arc (i,j),

$$wA_{ij} - c_{ij} = -w_i + w_j - c_{ij} = 0,$$

where $w_i$ and $w_j$ are termed the *node potentials* for node i and node j, respectively. Since B contains |N|-1 arcs and |N| nodes, it is necessary to solve a homogenous system of |N|-1 equations in |N| variables in order to determine the value of w. This system may be solved by first arbitrarily assigning any value to one of the variables and then solving the resulting system. This procedure also may be used to generate $\tilde{w}$ in order to obtain the $\lambda_k$'s of Step 3 (Section 2.0). The resulting value, $\tilde{w}_k$, associated with node k is called node k's *pseudo-node potential*. Although w and $\tilde{w}$ may be generated by using this same method, in actuality only $\tilde{w}$ is generated anew each iteration. Normally, the node potentials are computed at the beginning and then merely updated from one iteration to the next.

## 3.2 BASES AND SPANNING TREES

In the standard bounded variable simplex algorithm, a *basis B* for the CT problem is a matrix which consists of a linearly independent set of column vectors of A and which has one less than full row rank. The variables associated with the columns of B are considered to be basic variables and all others are nonbasic variables. Collectively, the vector of basic variables is denoted by $x_B$ while that of the nonbasic variables is denoted by $x_N$. A basic solution is obtained by assigning each nonbasic variable a value equal either to its upper bound or to zero (its lower bound). Hence, a unique value may be found for each basic variable which satisfies equation (2) of Lemke's dual method.

A basis for the CT problem may be viewed and stored as a graph which contains only the nonzero components of the matrix B [6, 12, 14, 16].

This basis graph contains all |N| nodes and exactly |N|- 1 arcs of the original problem graph. Since these arcs correspond to columns of B, they are linearly independent and the basis graph is thus a spanning tree. Henceforth, the terms *basis graph* and *spanning tree* will be used interchangeably. The node which is designated the *root node* will be viewed as the highest node in the spanning tree. (See Figure 2 for an example).

The next section briefly discusses the computer storage of the spanning tree corresponding to B and the original problem graph corresponding to A.

## 4.0 COMPUTER DATA STRUCTURES AND LABELING TECHNIQUES

### 4.1 STORING THE ORIGINAL PROBLEM GRAPH

Three different schemes for storing the original problem graph were implemented and evaluated to determine which scheme performs best.

The first scheme uses a popular linked structure [7,8] to store the original problem data as contained in the matrix A. In this method, all of the arcs which begin at the same node are stored together and each is represented by recording its ending node, costs, and upper bound. A pointer is then kept for each node; the pointer indicates the block of computer memory locations for the arcs beginning at this node. The set of arcs emanating from node u is called the forward star of node u and denoted by FS(u), i.e., FS(u) = $\{(u,j) \in E\}$. If the nodes are numbered sequentially from 1 to |N| and the arcs are stored consecutively in memory in such a way that the arcs in the forward star

of node i appear immediately after the arcs in the forward star of node i-1, this method, called the *forward star form*, requires |N| + 3|E| units of memory to store the original problem arc data [8]. These four arrays are referred to as the FROM POINTER, TO NODE, COST, and CAP arrays respectively. To complete the storage of the original problem data, one additional node length array, the NR array, is required; it stores the original node requirements for each node. Figure 1A illustrates a sample problem and Figure 1B shows its associated arrays.

The storage of the original problem data in consecutive memory locations using the forward star form implicitly associates a unique arc number with each arc. For each arc, this number identifies the position of the original problem data of that arc relative to the position of the original problem data of the first arc in the forward star of node 1 stored in memory. The uniqueness of this arc number readily allows the accommodation of multiple arcs between the same two nodes.

The other two schemes for storing the original problem data utilize the forward star form in conjunction with additional arrays which are designed to improve algorithmic performance. These approaches seek to lower problem solution time at the expense of extra storage requirements.

The second scheme utilizes the creation of two more arc length arrays and one node length array in order to identify easily all arcs entering the same node. One of the arc length arrays, the FROM NODE array, contains beginning node numbers which are blocked to indicate arcs

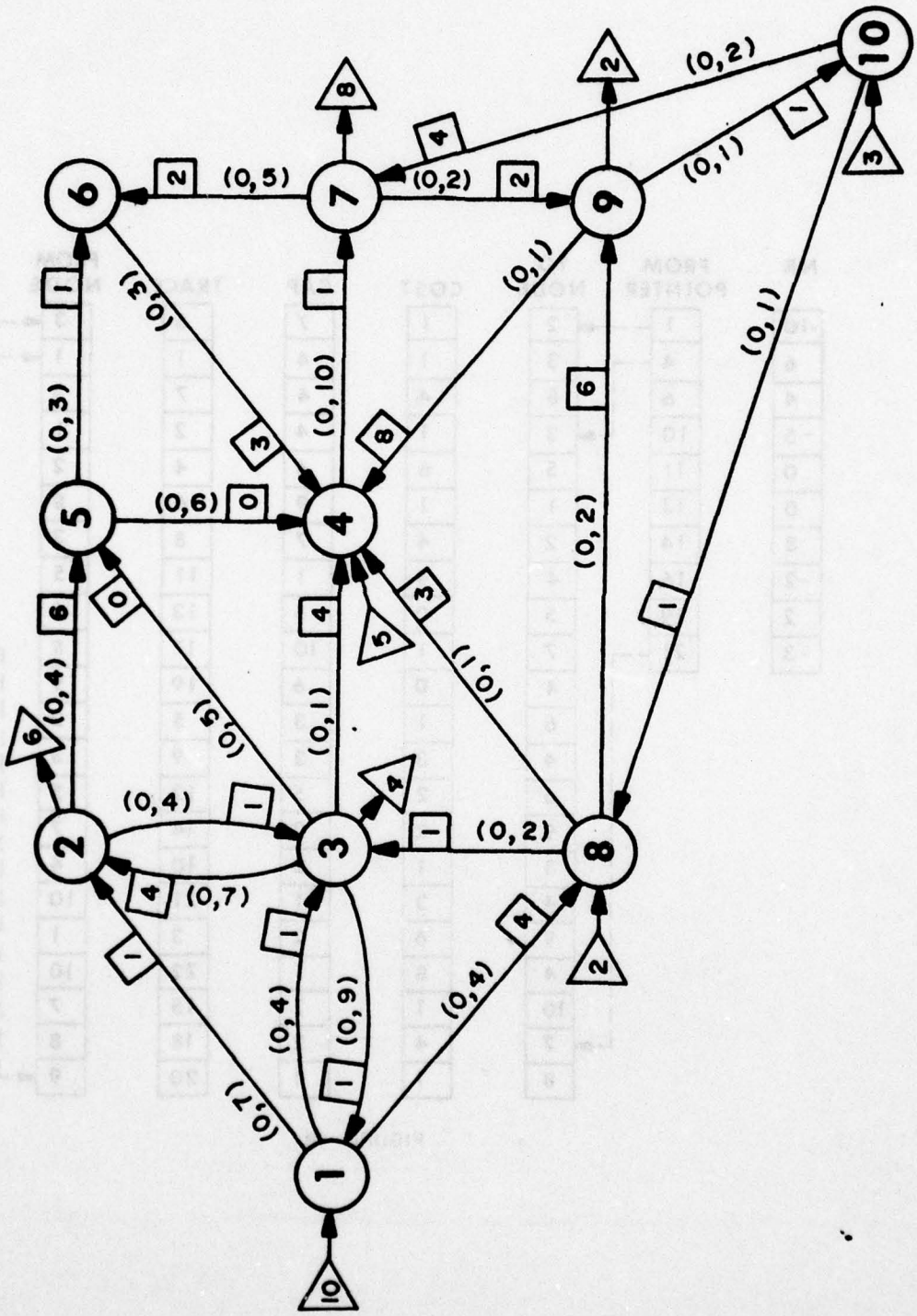

10
9
7
6
5
4
3
2
1
8
(0,2)
(0,1)
(0,5)
(0,2)
(0,3)
(0,10)
(0,1)
(0,6)
(0,2)
(0,1)
(0,4)
(0,5)
(0,1)
(1,0)
(0,4)
(0,2)
(0,7)
(0,4)
(0,9)
(0,4)
(0,7)

FIGURE 1A

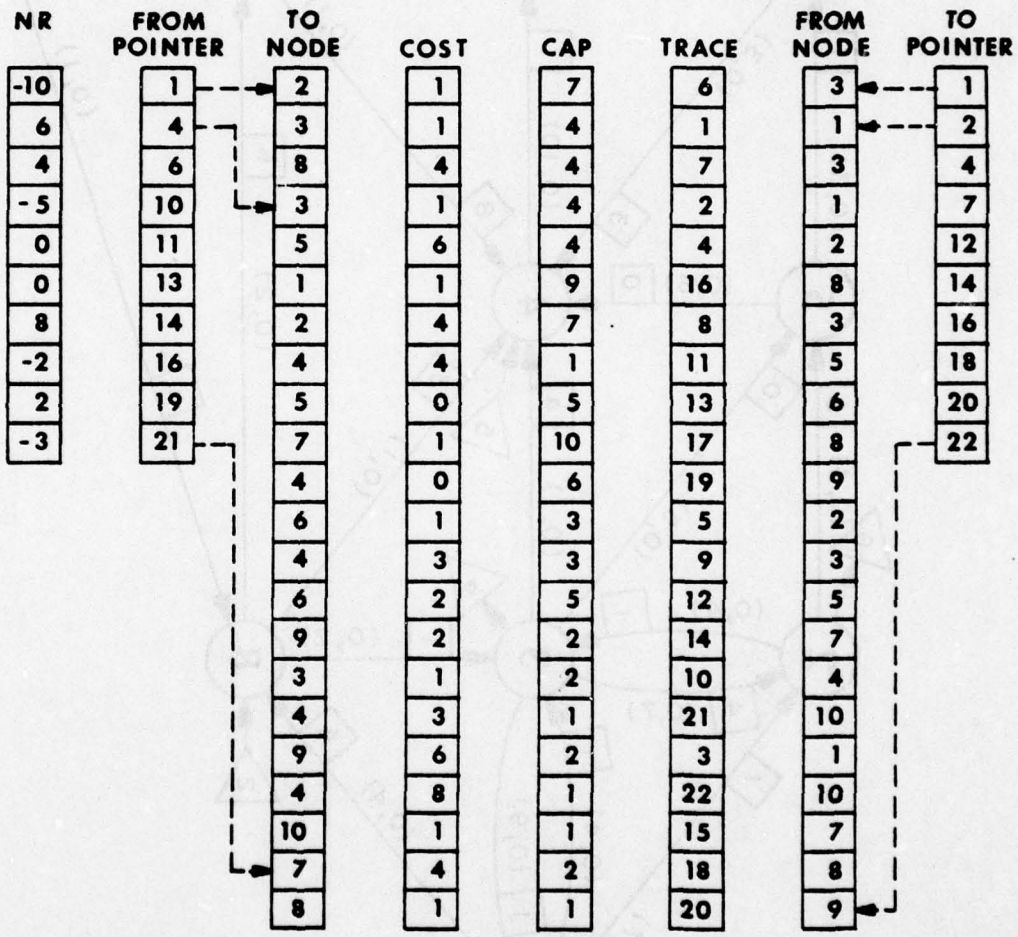

NR
FROM POINTER
TO NODE
COST
CAP
TRACE
FROM NODE
TO POINTER
-10 6 4 -5 0 0 8 -2 2 -3
1 4 6 10 11 13 14 16 19 21
2 3 8 3 5 1 2 4 5 7 4 6 4 6 9 3 4 9 4 10 7 8
1 1 4 1 6 1 4 4 0 1 0 1 3 2 2 1 3 6 8 1 4 1
7 4 4 4 4 9 7 1 5 10 6 3 3 5 2 2 1 2 1 1 2 1
6 1 7 2 4 16 8 11 13 17 19 5 9 12 14 10 21 3 22 15 18 20
3 1 3 1 2 8 3 5 6 8 9 2 3 5 7 4 10 1 10 7 8 9
1 2 4 7 12 14 16 18 20 22

FIGURE 18

having the same ending node. The node length array, the TO POINTER array, is used as an ending node index into this blocked set. Thus, this pointer system is a mirror image of the forward star form described above and shall be referred to as a *reverse star* form. The second arc length array, the TRACE array, allows convenient retrieval of the original cost and capacity data. These additional arrays are shown for the sample problem in Figure 1B.

The third scheme utilizes two extra $(2|N| + |E|)$ length arrays. The first $(2|N| + |E|)$ length array links together in two separate group's the nonbasic arcs at their upper bound and the nonbasis arcs at their lower bound of zero for each node. These pointer lists require an arc length array of storage. The additional node length arrays contain the arc number for the first element of each of these lists and are used to access these lists. The second $(2|N| + |E|)$ length array reverses the pointers of each list of the first array in order to facilitate pointer update.

## 4.2 STORING THE BASIS GRAPH

In order to represent a tree in a computer, one node, is designated as the root node, r. $P_{ij}$, for $i \neq j$, is then defined to be the unique path from node i to node j. If node j lies on $P_{ir}$, with $j \neq i$, then j is termed a *predecessor* of i and i is called a successor of j. Immediate predecessors and successors are endpoints of a common arc. The tree, therefore, may be represented by keeping a pointer list which contains the immediate predecessor of node i on $P_{ir}$ for each node $i \neq r$ in the tree. For convenience, the predecessor of the root node is

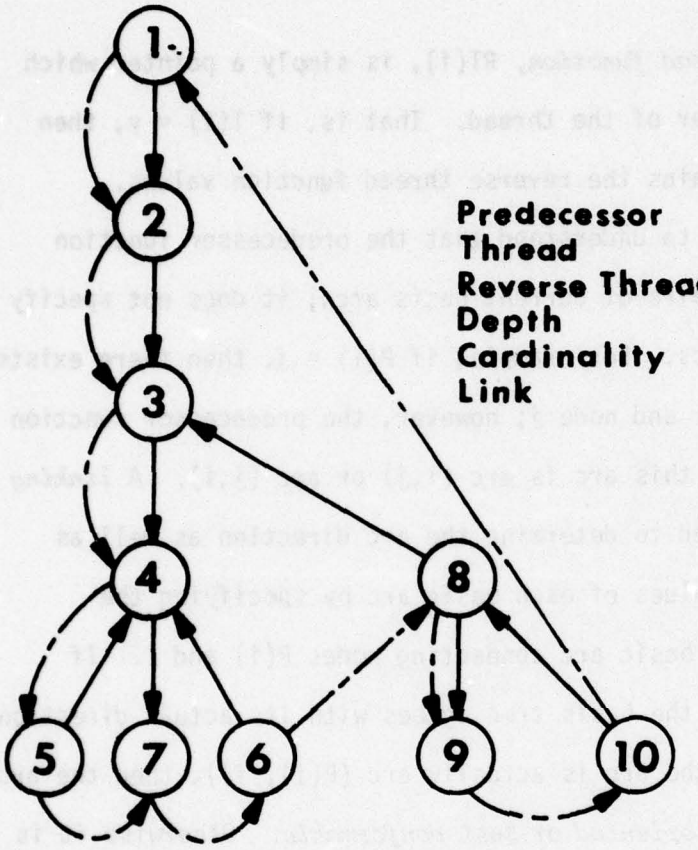
set equal to zero. Figure 2 illustrates a tree (for the network in Figure 1) rooted at node 1 and indicates the predecessors of the nodes as well as other functions which will be used in the computer implementation procedures to be discussed later in this paper. The predecessor of a node is identified in the P array. For example, the predecessor of node 8 is node 3.

The *thread function* [14], denoted T(i), is the first of these additional functions and is shown in Figure 2. It is used to facilitate the forward traversal of the spanning tree, an operation which is performed many times in the solution of CT problems by simplex based network codes. This function is a downward pointer through the tree and is illustrated in Figure 2 by the dotted line. Function T may be viewed as a connecting link (thread) which passes through each node exactly once in a top to bottom, left to right sequence, starting from the root node. For example, in Figure 2, T(1) = 2, T(4) = 5, T(10) = 1, etc.

The function T satisfies the following inductive characteristics:

(1) The set $\{r, T(r), T^2(r), \ldots, T^{N-1}(r)\}$ is precisely the set of nodes of the rooted tree, where by convention $T^2(r) = T(T(r))$, $T^3(r) = T(T^2(r))$, etc. The nodes $r, T(r), \ldots, T^{k-1}(r)$ are called the antecedents of node $T^k(r)$.

(2) For each node i other than node $T^{N-1}(r)$, T(i) is one of the nodes such that P(T(i)) = i, if such nodes exist. Otherwise, w will denote the first node which, in the predecessor path of i to the root, has an immediate successor y and for which y is not an antecedent of node i. In this case, T(i) = y.




| NODE | P | T | RT | DH | C | LINK |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 10 | 0 | 10 | — |
| 2 | 1 | 3 | 1 | 1 | 9 | 1 |
| 3 | 2 | 4 | 2 | 2 | 8 | 4 |
| 4 | 3 | 5 | 3 | 3 | 4 | 8 |
| 5 | 4 | 7 | 4 | 4 | 1 | 11 |
| 6 | 4 | 8 | 7 | 4 | 1 | 13 |
| 7 | 4 | 6 | 5 | 4 | 1 | 10 |
| 8 | 3 | 9 | 6 | 3 | 3 | 16 |
| 9 | 8 | 10 | 8 | 4 | 1 | 18 |
| 10 | 8 | 1 | 9 | 4 | 1 | 22 |

FIGURE 2

(3) $T^N(r) = r$; that is, the "last node" of the tree threads back to the root node.

The *reverse thread function*, RT(i), is simply a pointer which points in the reverse order of the thread. That is, if T(i) = y, then RT(y) = i. Figure 2 contains the reverse thread function values.

It is important to understand that the predecessor function only indicates the node pairs of current basis arcs; it does not specify the direction of these arcs. For example, if P(i) = j, then there exists a basic arc between node i and node j; however, the predecessor function does not indicate whether this arc is arc (i,j) or arc (j,i). A *linking function*, LINK (i), is used to determine the arc direction as well as the cost or upper bound values of each basic arc by specifying the network arc number of the basic arc connecting nodes P(i) and i. If this arc's orientation in the basis tree agrees with its actual direction in the network (i.e., if the arc is actually arc (P(i), i)), then the arc is said to be *conformably oriented* or just *conformable*. Otherwise it is said to be *nonconformably oriented* or simply *nonconformable*.

A final function is used in our implementation to facilitate the tree traversals, the efficient updating of the P, T, and RT functions, and the streamlining of the update of primal and dual solution data. Two functions (depth and cardinality) were considered for these purposes.

The *depth function*, DH(i), indicates the number of nodes in the predecessor path of node i to the root, disregarding the root node itself. If one imagines the nodes in the tree to be arranged in levels with the root at level zero, and all nodes "one node away from" the root at level one, etc., then the depth function simply indicates the level of a node in the tree. (See Figure 2.)

The *cardinality function*, C(i), specifies the number of nodes contained in the subtree associated with node i in the tree. By the nodes in the subtree associated with node i, we mean the set of all nodes $j \in N$ such that the predecessor path from j to the root contains i. (See Figure 2.)

For reasons which shall be explained in the following sections, the cardinality function is well suited to the implementations of the modified dual algorithm utilized in this study. Therefore, all codes discussed and tested in this paper utilize the cardinality rather than the depth function. For the sake of completeness, however, early testing compared the effect on solution time of the depth function versus the cardinality function. Codes differing solely in the implementations of these functions were run on the same problems. Uniformly those codes employing cardinality had lower total solutions times. For the sake of brevity, these results are not presented here but may be obtained from the authors.

Three additional node length arrays are used to store information pertaining to the basis. The first, the NP array, stores the current node potentials for the current basis. The second is a working array which alternately stores the basic arc flow values and pseudo-node potentials.

The third array, the NR array, was introduced in the discussion of the original problem data (Section 4.1). Initially, it is employed to store b, the original node requirements vector. In the context of a particular basis, the entries in the NR array are updated so that $\tilde{b} = b + N(-x_N)$; the node requirements vector, now "updated" with respect to the current basis, is stored. The term "updated" refers to the

modifications in the original node requirements (the original supply and demand requirements at each node) necessitated by the addition of $N(-x_N)$, which is precisely the negative sum of the arc flow on the nonbasic arcs at their upper bounds. More precisely, if the flow on the nonbasic arc (i,j) is at its upper bound for the initial dual feasible basis, then NR(i) is increased by $u_{ij}$ and NR(j) is decreased by $u_{ij}$. Whenever the flow on any nonbasic arc is changed via a multiple pivot operation, then the NR array must be updated.

The next section describes the specialization of the modified dual algorithm to the CT problem. The data structures used to represent the original problem data and the labeling functions used to represent the basis graph exert a powerful influence on the computational efficiency of the algorithm. A primary focus of this paper is to demonstrate the importance of the particular storage scheme used for the tree and network in executing the steps of the algorithm.

## 5.0 IMPLEMENTATION OF THE MODIFIED DUAL ALGORITHM IN NETWORKS

Section 5 synthesizes the ideas of the last two sections. In particular, it shows how the data structures and labeling techniques presented in the last section may be utilized to implement efficiently the modified dual algorithm for CT problems. The exposition follows the organization of the algorithm as presented in Section 2.

If an initial dual feasible basis B is known and is stored as a spanning tree using the predecessor, thread, reverse thread, and cardinality functions, and if the node potentials have been computed and stored in the NP array, then the initial primal solution $x_B$ now

may be determined in order to complete the first step of the algorithm.

## 5.1 COMPUTATION OF BASIC ARC FLOWS

The computation of $x_B$ requires solving the system $Bx_B = \tilde{b}$. Since B is represented by a spanning tree, and hence $Bx_B = \tilde{b}$ is a triangular system of equations, the system can be solved by back substitution. That is, it is necessary only to identify a row of B in which there is a single nonzero entry corresponding to a variable whose value has not yet been determined.

This solution mechanism is simple to implement using the RT, LINK, and P functions. The reverse thread of the root is a node i which has only one incident arc which connects nodes P(i) and i. The direction of this arc may be determined from the LINK array. Finally, the NR array may be used to assign the required flow on this arc in order to satisfy the updated node requirement for node i. By actually traversing the tree via the reverse thread, each node successively reached will have exactly one incident arc with a flow value to be determined. Thus, when the root node is reached, the solution $x_B$ is known.

It is important to note that the basic flows can be recomputed at any iteration by using the above procedure. When a single rather than a multiple pivot strategy is employed, it is more efficient to update the basic flows by finding the unique loop created in the basic tree when the entering arc is augmented to the tree. The flows on the basic arcs in this loop are then changed appropriately (See [10]). However, when the multiple pivot approach is employed, the flow values on several nonbasic arcs may change (from one bound to the opposite

bound for each arc). Accordingly, updating the basic flows necessitates finding and changing the flow on each of the basic arcs in the loop created by each nonbasic arc which changes its bound status as well as finding and changing the flow on basic arcs in the loop created by the entering arc. Hence we chose to update the NR array rather than a basic arc flow array and then recompute the values of the basic arcs as needed from one iteration to the next.

## 5.2 SELECTION OF THE LEAVING ARC

The second step of the dual method is to select a primal infeasible arc to leave the basis. (If all basic arcs are primal feasible, then the current solution is optimal and the procedure terminates.) Since any such arc may be chosen as the leaving arc, we tested four heuristic selection rules.

The first strategy selects the arc having the maximum flow infeasibility. The aim of this strategy is to find a "good" leaving arc so as to minimize, hopefully, the number of dual basis exchanges. However, to find the arc with the maximum flow infeasibility, all basis flow values must be computed and examined.

Since the basic arc flows are not updated from iteration to iteration but rather are recomputed at each iteration, it may be advantageous to devise pivot strategies which may not require all basic flows to be computed. Consequently, the second strategy selects the first primal infeasible arc encountered in the basic flows computation. While this strategy may result in more dual basis exchanges, all of the basic flows are rarely computed at each iteration.

The remaining two strategies are hybrids of the first two approaches. Strategies 3 and 4 recompute the basic flows until d (a positive integer) infeasible arc flows are encountered or all flows have been recomputed. Strategy 3 then selects as the leaving arc the one from this set of d arcs which has the most infeasible flow. Strategy 4 not only chooses the leaving arc in this manner but also resets the value of d to be e if e < d infeasible arcs were encountered.

The difference between Strategies 3 and 4 becomes pronounced if the number of infeasible basic arcs in the network decreases as a problem is solved. In such a case, strategy 3 approaches the most infeasible flow strategy while strategy 4 becomes the first infeasible flow strategy. Further, note that Strategies 3 and 4 with d = 1 are the same as the first infeasible flow strategy. (In the case of 4, if no infeasible arcs are encountered, the current solution is optimal; hence, the value of d is not reset.) Finally, Strategy 3 with d equal to the number of nodes is the same as the most infeasible flow strategy.

## 5.3 DETERMINATION OF THE ELEMENTS OF $NB^+$

Once arc (s,t) has been selected to leave the basis, the next step is to determine the elements of $NB^+$ via a three part procedure. First the pseudo-node potential values are calculated. Then $\lambda_{ik}$, the reduced pseudo cost value for each nonbasic arc (i,k), is computed. Finally, $NB^+$ is formed by evaluating $\lambda^1_{ik}$ for each nonbasic arc (i,k).

## 5.3.1 CALCULATION OF THE PSEUDO-NODE POTENTIALS

Calculating the pseudo-node potentials involves solving the system $\tilde{w}B = \tilde{c}_B$, where, as noted in section 3.1, one pseudo-node potential may be arbitrarily specified. The root node is customarily selected for this purpose and assigned a pseudo-node potential value of zero. In a naive implementation the pseudo-node potential values of the other nodes may then be determined in a cascading fashion. To do so, one moves down the tree via the thread function and distinguishes the pseudo-node potential value for each node i from that of its predecessor node j by using the equation $-\tilde{w}_i + \tilde{w}_j = \tilde{c}_{ij}$ for each basic arc, where $\tilde{c}_{st} = 1$ and $\tilde{c}_{ij} = 0$ for arc $(i,j) \in E - \{\text{arc } (s,t)\}$.

To improve the computational efficiency of the algorithm, this procedure should be streamlined in the following way. Arc (s,t) is removed from the tree. The subtree containing the root node is called the "main tree" and the other subtree is referred to as the "subordinate tree." Note that $-\tilde{w}_i + \tilde{w}_j = \tilde{c}_{ij} = 0$ for each arc (i,j) within each subtree. So $\tilde{w}_i = \tilde{w}_j$ and all nodes within the same subtree have the same pseudo-node potential value. Since $\tilde{w}_r = 0$, this imparts a zero pseudo-node potential value to every node in the main tree. The pseudo-node potential values of the nodes in the subordinate tree consequently depend on the direction of the leaving arc; that is, if arc (s,t) is conformable (nonconformable), then $\tilde{w}_s = 0$ ($\tilde{w}_t = 0$) and $-\tilde{w}_s + \tilde{w}_t = \tilde{c}_{st} = 1$ reduces to $\tilde{w}_t = +1$ ($\tilde{w}_s = -1$). Consequently, all of the pseudo-node potentials in the subordinate tree have a value of +1 if arc (s,t) is conformable and -1, otherwise. Hence, after the direction of the leaving arc has

been ascertained, all of the pseudo-node potential values may be set by traversing the two subtrees.

### 5.3.2 CALCULATION OF THE PSEUDO-REDUCED COSTS

The next step is to compute the pseudo-reduced cost, $\lambda_{ik}$, for each nonbasic arc (i,k). From Section 2.0,

$$\lambda_{ik} = \tilde{w}A_{ik} - c_{ik} = -\tilde{w}_i + \tilde{w}_k - \tilde{c}_{ik}.$$

If i and k are in the same subtree, $\lambda_{ik} = 0$, since $\tilde{w}_i = \tilde{w}_k$ and $\tilde{c}_{ik} = 0$. If i and k are nodes in different subtrees, then either $\tilde{w}_i$ or $\tilde{w}_k$ equals zero. Since $c_{ik} = 0$, the computation of $\lambda_{ik}$ for this case simplifies to the following: If arc (i,k) is directed from the main tree to the subordinate tree (from the subordinate tree to the main tree), the value of $\lambda_{ik}$ is the same as (the negative of) the value of the pseudo-node potential of any node in the subordinate tree. Since the values of the pseudo-node potentials depend only on the direction the leaving arc, the computation of the nonzero pseudo-reduced cost becomes solely a graphical consideration as shown in Table I.

TABLE I

| Direction of leaving arc with respect to predecessor orientation | Pseudo-reduced cost value | Nonbasic arcs From | Nonbasic arcs To |
|---|---|---|---|
| Nonconformable | +1 | subordinate tree | main tree |
| | -1 | main tree | subordinate tree |
| Conformable | +1 | main tree | subordinate tree |
| | -1 | subordinate tree | main tree |

Hence, the three pseudo-reduced cost values of 0 and ± 1 simply serve to partition the set of nonbasic arcs into three distinct sets. These three sets consist of nonbasic arcs between nodes in the same subtree, from nodes in the main tree to nodes in the subordinate tree, and from nodes in the subordinate tree to nodes in the main tree.

5.3.3 CALCULATION OF THE $\lambda_{ik}^{1}$'s

For each nonbasic arc (i,k) in each of these sets, $\lambda_{ik}^{1}$ may now be computed. This step completes the determination of $NB^{+}$ since

$$NB^{+} = \{(i,k) | \lambda_{ik}^{1} > 0 \text{ and } (i,k) \epsilon NB\}.$$

From Step 4 of Section 2.0,

$$\lambda^1_{ik} \begin{cases} \lambda_{ik} \text{ if } x_{st} < 0 \text{ and } x_{ik} = w_{ik} & (1) \\ \text{or } x_{st} > u_{st} \text{ and } x_{ik} = 0 & (2) \\ -\lambda_{ik} \text{ if } x_{st} < 0 \text{ and } x_{ik} = 0 & (3) \\ \text{or } x_{st} > u_{st} \text{ and } x_{ik} = w_{ik} & (4) \end{cases}$$

So, $\lambda^1_{ik} = \pm \lambda_{ik}$ depending on the values of the arc flow on arc (i,k) and on arc (s,t). Note that the nonbasic arcs between two nodes i and k in the same subtree are not eligible to enter the basis since $\lambda_{ik} = \lambda^1_{ik} = 0$. Further some nonbasic arcs in each of the other two sets may be eligible to enter the basis while other nonbasic arcs may not be. The only nonbasic arcs eligible to enter the basis are those arcs out of one subtree having a pseudo-reduced cost value of +1 and satisfying condition (1) or (2) and those arcs out of the second subtree which have a pseudo-reduced cost value of -1 and which satisfy condition (3) or (4).

A convenient way to establish the arcs to be included in $NB^+$ is to determine initially the flow status and arc direction of the leaving arc. After the arc direction is known, the pseudo-reduced cost values of the nonbasic arcs out of each subtree may be obtained from Table I. The flow status of arc (s,t) determines whether the nonbasic arcs indicated by equation 5.3's conditions (1) and (3) or (2) and (4) are then to be included in $NB^+$.

For example, if $x_{st} > u_{st}$ and arc (s,t) is nonconformable, then the nonbasic arcs from the subordinate tree to the main tree have a pseudo-reduced cost value of 1 (See Table I). Since $x_{st} > u_{st}$, Condition (2) indicates that of these nonbasic arcs only those at a zero flow level are eligible to enter the basis. Similarly, the nonbasic arcs from the main tree to the subordinate tree have a pseudo-reduced cost value of -1 and, of these arcs, Condition (4) dictates that only those arcs at their upper bounds are eligible to enter the basis. This case and the other three cases are listed in the following table.

TABLE II

| Leaving Arc (s,t) | | To Determine $NB^+$ Include Each Nonbasic Arc (i,k) out of | |
|---|---|---|---|
| Flow Status | Arc Direction | Main Tree | Subordinate Tree |
| $x_{st} > u_{st}$ | Nonconformable | At Upper Bound | At Zero Flow |
| $x_{st} < 0$ | Conformable | At Upper Bound | At Zero Flow |
| $x_{st} > u_{st}$ | Conformable | At Zero Flow | At Upper Bound |
| $x_{st} < 0$ | Nonconformable | At Zero Flow | At Upper Bound |

Neither the values of the pseudo-node potentials nor those of the pseudo reduced costs appear in Table II. Therefore, it is not necessary to compute them in order to determine the elements of $NB^+$. The procedure used to identify the elements included in $NB^+$ is as follows. First, the arc direction and flow status of the leaving arc are recorded. Secondly, the nodes of the subordinate tree are identified by flagging

their associated entries in the PNP array. Then appropriate nonbasic arcs are considered. In doing so, one must check whether or not the nodes i and k are in different subtrees and one must note the value of $x_{ik}$.

An ideal implementation would examine only the arcs between the two subtrees as presented in Table II. While this is not practical, alternative data structures may be designed to store the original problem data to reduce the number of arcs examined. Without these alternative structures, it is necessary to examine every arc, basic and nonbasic, in order to determine $NB^+$.

To serve as a basis for comparison, DNETA, the first code presented in Section 6.0, considers every arc in the forward star of each node. The second code, DNETD, implements the reverse star form in addition to the forward star form. After the cardinality function is used to determine the smaller subtree, each of the arcs into and each of the arcs out of the nodes of that subtree are checked to determine if they are basis eligible. Hence, fewer arcs are checked by DNETD than by DNETA. However, DNETD does consider nonbasis eligible arcs into and out of the smaller subtree a well as arcs between nodes within that subtree.

DNETL, the third code, uses the forward star form and partitions the nonbasic arcs out of each node into two doubly linked lists. The first list links together for each node the outgoing arcs at their upper bounds. In a similar fashion, the second list links together for each node the outgoing arcs at their lower bounds of zero. Consequently for

each subtree only the arcs out of each node at the flow values as specified by Table II are considered. However, arcs between nodes within the same subtree for both subtrees are examined.

It is impossible to predict á priori whether the second or third codes considers fewer arcs. Both methods consider some arcs which are nonbasis eligible and do not examine others. DNETD does not examine the basic arcs of the larger subtree while DNETL does not examine any basic arcs. Hence, DNETL has a "better" worst case bound. However, DNETD only processes the arcs in the forward and reverse stars of the nodes in the smaller subtree, while DNETL considers the forward star of every node in the tree. Considering the examination of each arc in the forward and the reverse stars of one node as two "node scans", DNETD, on the average, scans fewer nodes. But, since it is theoretically impossible to predict whether DNETD or DNETL is more efficient, both codes were tested in conjunction with DNETA. DNETD proved superior to both DNETL and DNETA; the results appear in Section 6.3.

The examination of the nonbasic arcs is the only difference in the three codes. All three implementations actually store the necessary nonbasic arcs required to perform the multiple pivot in the same manner. Obviously, the particular storage scheme used (to be discussed in the next section) affects the ease of the calculation of the entering arc, which is the next step of the modified dual method.

## 5.4 SELECTION OF THE ENTERING ARC

To determine the entering arc, it is necessary to compute $|\pi_{ij}/\lambda_{ij}|$ for each $(i,j)\epsilon NB^+$. Since $|\lambda_{ij}| = 1$ for all $(i,j)\epsilon NB^+$, there

is no ratio to compute and this calculation reduces to

$$|\pi_{ij}/\lambda_{ij}| = |\pi_{ij}| = |-w_i + w_j - c_{ij}|$$

This is straight forward once $NB^+$ has been determined.

For the purpose of exposition, suppose that all $|\pi_{ij}|$'s with $(i,j)\epsilon NB^+$ have been sorted in ascending order. Let the r-th position in this list be such that the summation of the $u_{ij}$'s associated with the first r-1 entries is less than $I_{st}$ and the summation of the $u_{ij}$'s associated with the first r entries is greater than or equal to $I_{st}$. Let $|\pi_{pq}|$ be this r-th entry; then arc (p,q) is identified to enter the basis. By theorem 1, the solution remains dual feasible after the basis exchange takes place.

The modified dual algorithm can be implemented using the complete sort as described above. However, to perform a modified dual pivot one need know only which arc (p,q) will enter the basis and which arcs (i,j) correspond to the first r-1 entries described above in the complete sort. A more efficient partial sort may be implemented by defining NB* to be the subset of $NB^+$ associated with these r entries. Initially define arc (p,q) to be an artificial basic eligible arc with a large positive reduced cost and upper bound. $NB^*$ is built and (p,q) updated by comparing $|\pi_{ij}|$, $(i,j)\epsilon NB^+$, against $|\pi_{pq}|$ and either discarding it if $|\pi_{ij}| \geq |\pi_{pq}|$, or redetermining (p,q) and $NB^*$ if $|\pi_{ij}| < |\pi_{pq}|$. Computationally, $NB^+$ and the partially sorted list are computed simultaneously by making one pass through the appropriate subset of the nonbasic arcs.

Three "sort" arrays store the elements of $NB^*$ by recording the arc number, the FROM node, and the absolute value of the reduced cost (ratio value) of each arc corresponding to an index contained in $NB^*$. The length of these arrays must be $|E|-|N| + 1$ in order to implement the modified dual

algorithm. The length of these arrays in DNETA, DNETD, and DNETL were accordingly set to this size.

If the length of these arrays is reduced to some number $K < |E|-|N|+1$, then at most the K smallest minimum ratios will be retained. What emerges is a hybrid approach utilizing a combination of a single and multiple pivot strategy. If K is less than the cardinality of $NB^*$, a nonbasic variable corresponding to the largest (in absolute value) ratio in the sort list enters the basis at a primal infeasible level. Since DNETD proved to be the computationally superior code, it was modified to incorporate this hybrid approach and tested in concert with DNETD. The results appear in Section 6.5.

## 5.5 PERFORMING THE BASIS EXCHANGE

The last step of the modified dual algorithm is the basis exchange. The leaving vector becomes nonbasic at the bound it previously violated, the entering arc becomes basic, and the flow values of the nonbasic arcs corresponding to the other indices in $NB^*$ (if any) switch from their current bounds to their other bounds.

The update is performed in the following manner. First, the thread function which was cut in order to determine the elements of $NB^*$, is restored to its previous state. Then the basis exchange is performed and the thread is updated as discussed in [ 14 ]. Simultaneously the node potentials and the functions used to define the spanning tree are updated. Finally, the NR array is updated: if the flow on the nonbasic arc (i,j) is to be increased (decreased) to its other flow bound, then NR(i) is increased (decreased) by $u_{ij}$ and NR(j) is decreased (increased) by $u_{ij}$.

With the updated NR array and the updated basis, the basic arc flows are recomputed as required by the pivot selection procedure used. The procedure then returns to Step 2.

## 5.6 AN EXAMPLE OF A MODIFIED DUAL PIVOT

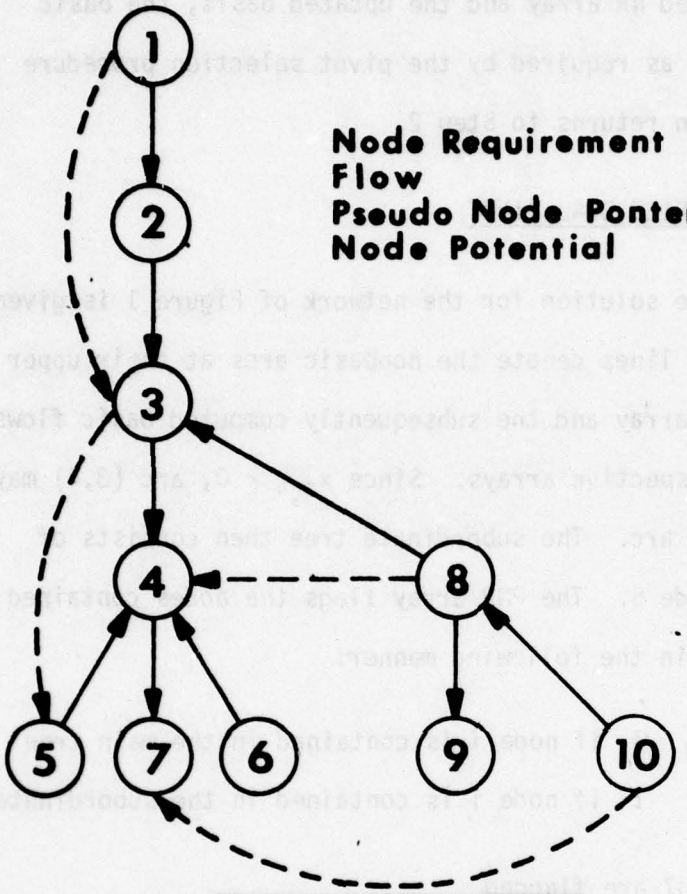
A dual feasible solution for the network of Figure 1 is given in Figure 3. The dotted lines denote the nonbasic arcs at their upper bounds. The updated NR array and the subsequently computed basic flows are recorded in their respective arrays. Since $x_{3,4} < 0$, arc (3,4) may be chosen as the leaving arc. The subordinate tree then consists of the subtree rooted at node 5. The PNP array flags the nodes contained in the subordinate tree in the following manner:

$$\text{PNP}(i) = \begin{cases} 0 & \text{if node } i \text{ is contained in the main tree} \\ 1 & \text{if node } i \text{ is contained in the subordinate tree} \end{cases}$$

Hence, nodes 4, 5, 6 and 7 are flagged.

Since $x_{3,4} < 0$ and arc (3,4) is oriented away from the root, it is only necessary to consider nonbasic arcs out of the main (subordinate) tree at the upper bound (zero flow level). Hence, only the absolute values of the reduced costs of arcs (3,5), (8,4), and (10,7), which are out of the main tree at their upper bounds, and that of arc (7,9), which is the only nonbasic arc out of the subordinate tree into the main tree at a zero flow level, are considered. Now, using the NP array together with the relevant arc costs,

$$|\pi_{3,5}| = 4,\ |\pi_{8,4}| = 2,\ |\pi_{10,7}| = 3,\ |\pi_{7,9}| = 2 \text{ and } I_{3,4} = 5.$$




| | |
|---|---|
| Node Requirement | NR (i) |
| Flow | FLOW (i) |
| Pseudo Node Pontential | PNP (i) |
| Node Potential | NP (i) |

| NODE | NR | FLOW | PNP | NP |
|---|---|---|---|---|
| 1 | -6 | — | 0 | 0 |
| 2 | 6 | 6 | 0 | 1 |
| 3 | 5 | 0 | 0 | 2 |
| 4 | -6 | -5 | 1 | 6 |
| 5 | -5 | 5 | 1 | 6 |
| 6 | 0 | 0 | 1 | 3 |
| 7 | 6 | 6 | 1 | 7 |
| 8 | -1 | 0 | 0 | 1 |
| 9 | 2 | 2 | 0 | 7 |
| 10 | -1 | 1 | 0 | 0 |

FIGURE 3

Since $u_{8,4} = 1$, $u_{7,9} = 2$, and $u_{10,7} = 2$, $u_{8,4} + u_{7,9} + u_{10,7} = I_{3,4}$.

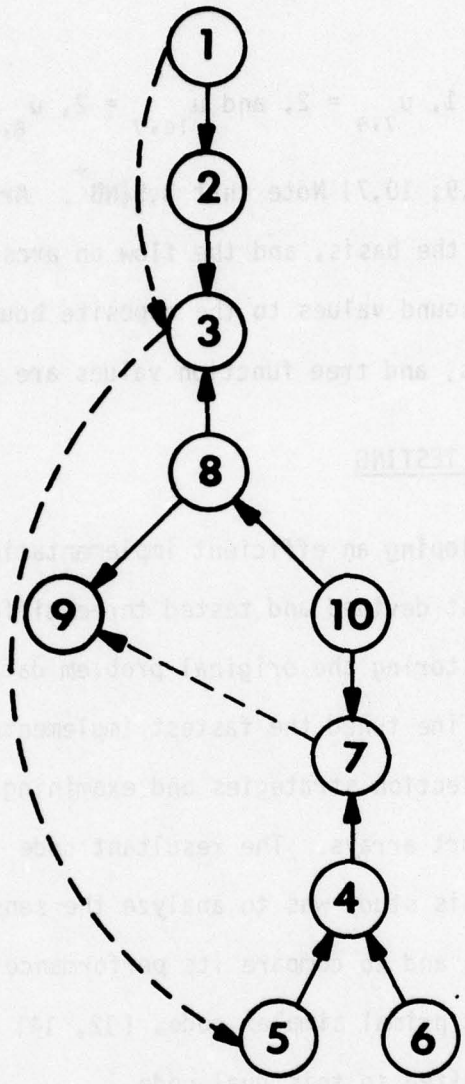
So, $NB^* = \{8,4;\ 7,9;\ 10,7\}$ Note that $3,5 \notin NB^*$. Arc (3,4) leaves the basis, arc (10,7) enters the basis, and the flow on arcs (8,4) and (7,9) switches from the current bound values to the opposite bound values. The updated NR array, flow values, and tree function values are shown in Figure 4.

## 6.0 COMPUTATIONAL TESTING

In developing an efficient implementation of the multiple pivot algorithm, we first devised and tested three different codes based on three methods of storing the original problem data as presented in Section 4.1. We fine tuned the fastest implementation by evaluating a set of pivot selection strategies and examining the effect of restricting the size of the sort arrays. The resultant code is called DNET2. The final aspect of this study was to analyze the sensitivity of DNET2 to problem parameters and to compare its performance to that of PNET-I, a state-of-the-art primal simplex code, [12, 14] over a set of problems especially well suited to this dual code.

### 6.1 CODE DEVELOPMENT

All of the codes, primal and dual, used in this study are in-core codes; that is, the program and all of the problem data simultaneously reside in fast-access memory. They are all coded in FORTRAN and none of them has been designed for a particular compiler. All of the problems were solved on the CDC 6600 at the University of Texas at Austin Computational Center using the MNF compiler. The




| NODE | NR | FLOW | P | T | RT | DH | C | LINK |
|---|---|---|---|---|---|---|---|---|
| 1 | -6 | — | 0 | 2 | 6 | 0 | 10 | — |
| 2 | 6 | 6 | 1 | 3 | 1 | 1 | 9 | 1 |
| 3 | 5 | 0 | 2 | 8 | 2 | 2 | 8 | 4 |
| 4 | -5 | 10 | 7 | 5 | 7 | 6 | 3 | 10 |
| 5 | -5 | 5 | 4 | 6 | 4 | 7 | 1 | 11 |
| 6 | 0 | 0 | 4 | 1 | 5 | 7 | 1 | 13 |
| 7 | 8 | -2 | 10 | 4 | 10 | 5 | 4 | 21 |
| 8 | -2 | 5 | 3 | 9 | 3 | 3 | 7 | 16 |
| 9 | 0 | 0 | 8 | 10 | 8 | 4 | 1 | 18 |
| 10 | -1 | 3 | 8 | 7 | 9 | 4 | 5 | 22 |

FIGURE 4

computer jobs were executed during periods when the machine load was approximately the same, and all solution times are exclusive of input and output; that is, the total time spent solving the problem was recorded by calling a Real Time Clock upon starting on the problem solution and again when the solution was obtained.

All of the dual codes used in this study employed the same starting procedure, the Advanced Dual Start [15], and utilized the partially sorted list of Section 5.4 in conjunction with the multiple pivot strategy.

## 6.2 PROBLEM SET DESCRIPTION

Before conducting the computational testing phase of this project, we postulated that the dual codes would perform well on tightly capacitated networks in which the ratio of arcs to nodes is high and the flows on a substantial number of the nonbasic arcs (20% or more) would be at their upper bounds at optimality. We theorized that for such networks with large node requirements and tight arc flow capacities the average number of elements in the sort lists per iteration would be large and that therefore, on the average, the flow on many nonbasic arcs would swing from one bound to the other during an iteration. Hence, such networks would make full use of the multiple pivot strategy.

A problem set of 22 test problems was selected to test this hypothesis. Generated by the NETGEN network generator [18], this problem set consists of 10 transportation and 12 transshipment networks.

[The number of nodes considered varies from 100 to 400 per network and the number of arcs varies from 274 to 5000.] All of the 22 networks have cost ranges of from 1 to 100, and each network has a total supply equal to 100 times the number of nodes in the network. Table III describes the 22 test problems. For some of these problems, certain parameters were modified so that sensitivity analysis could be performed. These parameters include number of arcs, cost range, total supply, and upper bound values. This second set encompasses 15 additional problems and is described in Table IV.

## 6.3 STORAGE OF THE ORIGINAL PROBLEM DATA

The first phase of testing evaluated the different methods of storing the original problem data. A separate code was written for each scheme; they are called DNETA, DNETL, and DNETD. DNETA requires 9 node length arrays (FROM, P, T, RT, LINK, C, NP, NR, and one working array), 3 arc length arrays (TO, COST, and CAP arrays) and 3 arrays which are used to store the data required to perform the partial sort to obtain $NB^*$. These sort arrays record the arc number, the FROM node, and the absolute value of the reduced cost of each arc corresponding to an index contained in $NB^*$. The dimension of each sort array is $|E| - |N| + 1$. DNETL uses the structures of DNETA in conjunction with a doubly-linked list which, for each node, ties together the nonbasic arcs out of that node at the same bound, either the upper or zero bound. In addition to the structures of DNETA, DNETD uses the reverse star structures which link together the arcs entering each node. The number and types of arrays required for each code are indicated in Table V.

TABLE III

TRANSPORTATION

| PROBLEM NUMBER | TOTAL NUMBER OF NODES | NUMBER OF SOURCES | NUMBER OF SINKS | NUMBER OF ARCS | COST RANGE | TOTAL SUPPLY | HI COST | CAP | UPPER BOUND MIN | UPPER BOUND MAX | SEED |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 100 | 5 | 95 | 250 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 |
| 2 | 100 | 10 | 90 | 300 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 |
| 3 | 100 | 10 | 90 | 1000 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 |
| 4 | 100 | 25 | 75 | 2000 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 |
| 5 | 100 | 50 | 50 | 2500 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 |
| 6 | 200 | 20 | 180 | 1200 | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 |
| 7 | 200 | 20 | 180 | 2200 | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 |
| 8 | 200 | 50 | 150 | 5000 | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 |
| 9 | 300 | 10 | 290 | 1000 | 1-100 | 30,000 | 80 | 100 | 5 | 10 | 12345678 |
| 10 | 400 | 40 | 360 | 5000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 |

TRANSSHIPMENT

| PROBLEM NUMBER | TOTAL NUMBER OF NODES | NUMBER OF SOURCES | NUMBER OF SINKS | NUMBER OF ARCS | COST RANGE | TOTAL SUPPLY | HI COST | CAP | UPPER BOUND MIN | UPPER BOUND MAX | SEED | TRANSSHIPMENT SOURCES | TRANSSHIPMENT SINKS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 100 | 7 | 50 | 1600 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 11A | 100 | 7 | 50 | 2000 | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 12 | 200 | 7 | 170 | 1600 | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 12A | 200 | 7 | 170 | 2000 | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 13 | 400 | 7 | 370 | 2000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 13A | 400 | 7 | 370 | 2500 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14 | 400 | 10 | 360 | 2000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14A | 400 | 10 | 360 | 3000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 15 | 400 | 10 | 350 | 3000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 16 | 400 | 200 | 350 | 2000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 16A | 400 | 20 | 350 | 3000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 17 | 400 | 10 | 360 | 4000 | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |

TABLE IV

SENSITIVITY ANALYSIS TESTING

| TOTAL NUMBER OF NODES | # SOURCES | # SINKS | # ARCS | ARC DENSITY | COST RANGE | TOTAL SUPPLY | HI COST | CAP | UPPER BOUND MIN | UPPER BOUND MAX | SEED | TRANSSHIPMENT SOURCES | TRANSSHIPMENT SINKS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 100 | 25 | 75 | 2000 | | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 4E 100 | 25 | 75 | 2000 | | 1-1000 | 10,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 9 300 | 10 | 290 | 1000 | | 1-100 | 30,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 9E 300 | 10 | 290 | 1000 | | 1-1000 | 30,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 11 100 | 7 | 50 | 1600 | | 1-100 | 10,000 | 80 | 10 | 5 | 10 | 12345678 | - | - |
| 11A 100 | 7 | 50 | 2000 | | 1-100 | 10,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 11B 100 | 7 | 50 | 1600 | | 1-100 | 1,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 11C 100 | 7 | 50 | 1600 | | 1-100 | 10,000 | 80 | 100 | 50 | 100 | 12345678 | - | - |
| 11D 100 | 7 | 50 | 1600 | | 1-100 | 1,000 | 80 | 100 | 50 | 100 | 12345678 | - | - |
| 11E 100 | 7 | 50 | 1600 | | 1-1000 | 10,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 12 200 | 7 | 170 | 1600 | | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 12A 200 | 7 | 170 | 2000 | | 1-100 | 20,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 12B 200 | 7 | 170 | 1600 | | 1-100 | 2,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 12C 200 | 7 | 170 | 1600 | | 1-100 | 2,000 | 80 | 100 | 50 | 100 | 12345678 | - | - |
| 13 400 | 7 | 370 | 2000 | | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 13A 400 | 7 | 370 | 2500 | | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 13B 400 | 7 | 370 | 2000 | | 1-100 | 4,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 13C 400 | 7 | 370 | 2000 | | 1-100 | 4,000 | 80 | 100 | 50 | 100 | 12345678 | - | - |
| 13E 400 | 7 | 370 | 2000 | | 1-1000 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14 400 | 10 | 360 | 2000 | | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14B 400 | 10 | 360 | 2000 | | 1-100 | 4,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14C 400 | 10 | 360 | 2000 | | 1-100 | 4,000 | 80 | 100 | 50 | 100 | 12345678 | - | - |
| 14A 400 | 10 | 360 | 3000 | | 1-100 | 40,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14A.1 400 | 10 | 360 | 3000 | | 1-100 | 4,000 | 80 | 100 | 5 | 10 | 12345678 | - | - |
| 14A.2 400 | 10 | 360 | 3000 | | 1-100 | 4,000 | 80 | 100 | 50 | 100 | 12345678 | - | - |

TABLE V

| CODE | ARRAY REQUIREMENTS REQUIRED TO PERFORM THE ALGORITHM | EXTRA ARRAYS REQUIRED FOR ADVANCED DUAL START | TOTAL ARRAY REQUIREMENTS |
|---|---|---|---|
| DNETA | 9\|N\|+ 3\|E\|+ 3\|V\| | 2\|N\|+ 1\|E\| | 11\|N\|+ 4\|E\|+ 3\|V\| |
| DNETL | 13\|N\|+ 5\|E\|+ 3\|V\| | NONE | 13\|N\|+ 5\|E\|+ 3\|V\| |
| DNETD | 10\|N\|+ 5\|E\|+ 3\|V\| | 1\|N\| | 11\|N\|+ 5\|E\|+ 3\|V\| |

where $|N|$ = number of nodes

$|E|$ = number of arcs

$|V| = |E|-|N|+ 1$

The three codes were compared by solving each of the 22 original test problems twice, using a different pivot change criterion each time. The first negative criterion computes the flow on each basic arc until a basic arc is encountered which violates either its zero or lower bound. That arc is then selected to leave the basis. The most negative criterion computes the flow on each basic arc and picks to leave the basis that arc which violates either its upper or lower bound by the largest amount. The first negative criterion was determined to be superior in an earlier study [15] in which the test problems were considerably different. The networks of that problem set had fewer arcs, the capacities on the arcs were considerably higher, and the node requirements were smaller. Consequently, in order to evaluate thoroughly the storage schemes, the most infeasible criterion was also used.

Without exception, for either pivot selection criterion employed, the results in Tables VI and VII indicate DNETD runs faster than DNETL, which runs faster than DNETA.

TABLE VI

FIRST NEGATIVE PIVOT SELECTION CRITERION

CP TIME IN SECONDS ON CDC 6600

| | TOTAL SOLUTION TIME | | | NUMBER OF PIVOTS | | |
|---|---|---|---|---|---|---|
| PROBLEM | DNETA | DNETL | DNETD | DNETA | DNETL | DNETD |
| 1 | .968 | .746 | .568 | 92 | 96 | 91 |
| 2 | 1.154 | .848 | .637 | 114 | 114 | 110 |
| 3 | 6.679 | 4.029 | 2.517 | 433 | 395 | 360 |
| 4 | 17.842 | 11.243 | 4.986 | 686 | 690 | 682 |
| 5 | 20.779 | 11.353 | 7.610 | 690 | 638 | 999 |
| 6 | 10.818 | 7.277 | 4.413 | 431 | 423 | 433 |
| 7 | 20.285 | 11.913 | 6.703 | 546 | 464 | 476 |
| 8 | 58.333 | 38.421 | 14.422 | 889 | 932 | 990 |
| 9 | 9.835 | 7.136 | 5.122 | 321 | 317 | 322 |
| 10 | 93.582 | 60.759 | 36.486 | 1222 | 1262 | 1932 |
| 11 | 20.970 | 13.954 | 6.205 | 866 | 904 | 847 |
| 11A | 32.457 | 21.546 | 8.531 | 1137 | 1176 | 1222 |
| 12 | 20.613 | 13.399 | 10.621 | 720 | 690 | 886 |
| 12A | 28.369 | 17.575 | 12.767 | 838 | 799 | 894 |
| 13 | 53.893 | 36.191 | 23.943 | 1345 | 1404 | 1435 |
| 13A | 71.214 | 43.382 | 33.955 | 1638 | 1534 | 1957 |
| 14 | 42.966 | 30.099 | 15.957 | 897 | 961 | 988 |
| 14A | 60.879 | 41.914 | 22.590 | 1027 | 1042 | 1215 |
| 15 | 39.478 | 26.379 | 15.848 | 874 | 873 | 932 |
| 16 | 30.365 | 21.275 | 15.419 | 697 | 721 | 894 |
| 16A | 56.077 | 40.004 | 20.412 | 1022 | 1078 | 1166 |
| 17 | 85.682 | 59.541 | 25.471 | 1216 | 1317 | 1190 |

TABLE VII

MOST NEGATIVE PIVOT SELECTION CRITERION

CP TIME IN SECONDS ON CDC 6600

| | TOTAL SOLUTION TIME | | | NUMBER OF ITERATIONS | | |
|---|---|---|---|---|---|---|
| PROBLEM | DNETA | DNETL | DNETD | DNETA | DNETL | DNETD |
| 1 | 1.117 | .869 | .808 | 91 | 91 | 91 |
| 2 | 1.416 | 1.127 | 1.023 | 105 | 109 | 104 |
| 3 | 3.476 | 2.835 | 2.343 | 128 | 147 | 139 |
| 4 | 8.868 | 5.871 | 5.219 | 203 | 190 | 198 |
| 5 | 14.006 | 11.677 | 9.106 | 291 | 331 | 311 |
| 6 | 10.104 | 7.351 | 6.415 | 279 | 273 | 271 |
| 7 | 16.785 | 12.150 | 10.432 | 298 | 296 | 316 |
| 8 | 58.313 | 41.786 | 32.442 | 617 | 594 | 589 |
| 9 | 12.573 | 9.439 | 8.604 | 328 | 340 | 343 |
| 10 | 87.814 | 65.309 | 52.604 | 783 | 825 | 808 |
| 11 | 8.259 | 5.475 | 5.139 | 184 | 160 | 170 |
| 11A | 8.564 | 7.085 | 5.390 | 188 | 189 | 183 |
| 12 | 11.003 | 7.940 | 7.131 | 244 | 241 | 247 |
| 12A | 13.144 | 9.572 | 7.932 | 259 | 252 | 253 |
| 13 | 29.036 | 21.330 | 18.008 | 481 | 481 | 456 |
| 13A | 33.818 | 28.118 | 20.906 | 468 | 548 | 472 |
| 14 | 33.015 | 24.810 | 20.010 | 500 | 508 | 477 |
| 14A | 49.751 | 33.540 | 27.017 | 707 | 515 | 513 |
| 15 | 32.210 | 23.932 | 21.301 | 500 | 489 | 512 |
| 16 | 32.851 | 23.975 | 20.204 | 521 | 525 | 542 |
| 16A | 49.674 | 37.099 | 28.850 | 619 | 628 | 613 |
| 17 | 58.529 | 44.534 | 34.813 | 556 | 576 | 574 |

The codes differ only in their implementation of the "arc scan", the order in which nonbasic arcs are examined to determine the arc to enter the basis. DNETA examines every nonbasic arc to determine NB*. For each arc, it must be determined if its two nodes are in different subtrees, and, if so, whether the flow on the arc is directed toward or away from the root, and whether the flow on the arc is at the upper or lower bound. If the arc then is determined to be eligible to enter the basis, its reduced cost is computed. If the absolute value of the reduced cost is less than that of the largest element currently in NB*, this arc is placed in NB*. The time consuming character of these operations is indicated by the lengthy solution times. DNETL eliminates by way of the data structure the arc capacity check and the processing of the current basic arcs; as Tables VI and VII show, the total solution times all decrease. DNETD reduces the amount of work in a different manner. It processes only the arcs in and out of the smaller subtree. The same number of operations per arc are performed as in DNETA but the forward and reverse stars of fewer nodes are examined. This approach can dramatically reduce the number of arcs processed; the extent of this reduction is indicated by the clear dominance of DNETD on every problem.

## 6.4 EVALUATION OF PIVOT SELECTION RULES

Neither DNETD/MN (DNETD with the most negative criterion) nor DNETD/FN (DNETD with the first negative criterion) was clearly superior over the entire problem set. DNETD/FN ran faster on 9 of the 10 transportation problems and on 6 of the 12 transshipment problems. Further,

the solution times for several of the problems indicate a strong dependence upon the pivot selection rule used. For example, DNETD/MN required 32.442 seconds to solve problem 8 while DNETD/FN required only 14.422 seconds. On the other hand, DNETD/MN solved problem 13A in 20.906 seconds while DNETD/FN needed 33.955 seconds to reach optimality.

In addition to these two pivot criteria, we investigated alternative pivot criteria and compared the performance of DNETD with each pivot selection criterion. The alternative criteria examined are based on Strategies 3 and 4 of Section 5.2. These strategies choose as the entering arc the most infeasible of the first d infeasible basic arcs encountered in the basic flow update. In preliminary testing, the reset strategy, strategy 4, proved to be uniformly better and, thus, strategy 3 was subsequently dropped.

In implementing strategy 4, two rules were used in determining value for d. First, d was set to be constant for all problem sizes. The values chosen were 10 and 40. (The versions of DNETD employing these criteria are referred to as DNETD/10 and DNETD/40, respectively.) Secondly, d was set to be a percentage of the number of nodes. The values examined were 5%, 10%, 20%, and 100%. (The versions of DNETD employing these rules are referred to as DNETD/5%, DNETD/10%, and DNETD/20%, and DNETD/100%.) Thus six versions of DNETD employing strategy 4 were evaluated in conjunction with DNETD/FN and DNETD/MN.

Problem Set 2, the problem set employed for this testing, includes 7 of the original 22 problems and 1 additional transshipment problem (taken from the 15 additional problems shown in Table IV) with smaller node requirements. Two of the eight problems are transportation

problems. Each "unique"[1] version of DNETD solved these eight problems twice. First, the problems were solved in order to compare total solution times and number of pivots. These statistics appear in Table VIII. Then counters were inserted in each version. For each pivot two counts were made, one of the number of nodes in the smaller subtree and the other of the number of basic arcs whose flows were recomputed in order to find the leaving arc. Both counts were accumulated and are also shown in Table VIII. Note that the number of nodes in the smaller subtree is the number of nodes whose forward and reverse stars must be scanned in order to determine NB*.

Viewing the data of this table, one can see that in general a code implementing a strategy which scans the forward and reverse stars of fewer nodes will tend to solve problems faster. However, that same code may process more arcs, may require many more pivots, or may require that the flow on more basic arcs be computed in order to determine NB*. Problem 4 serves as a good example. DNETD/MN scans fewer forward and reverse stars than does DNETD/10%. However, DNETD/MN must compute more basic flows.

---

[1]For certain problem sizes two versions of DNETD are identical. For example, for 400 node networks, DNETD/10% and DNETD/40 are the same. Consequently, for such situations only one run was performed for both versions and the same information was recorded for both versions.

TABLE VIII

LEAVING ARC SELECTION STRATEGIES

CP TIME IN SECONDS ON CDC 6600

| STRATEGY \ PROBLEMS | 4 | 9 | 11A | 12 | 14 | 14A | 14A.1 | 17 |
|---|---|---|---|---|---|---|---|---|
| **TOTAL SOLUTION TIME** | | | | | | | | |
| DNETD/MN | 5.219 | 8.604 | 5.390 | 7.131 | 20.010 | 31.107 | 27.017 | 34.401 |
| DNETD/FN | 4.916 | 5.141 | 8.469 | 10.596 | 15.930 | 22.722 | 22.584 | 25.401 |
| DNETD/10 | 5.043 | 5.096 | 7.363 | 7.934 | 15.487 | 21.506 | 23.456 | 24.803 |
| DNETD/40 | 5.683 | 5.487 | 7.747 | 7.734 | 14.320 | 21.168 | 20.946 | 24.797 |
| DNETD/100% | 4.731 | 8.904 | 5.852 | 6.953 | 20.961 | 31.812 | 28.142 | 35.354 |
| DNETD/5% | 5.357 | 5.661 | 8.576 | 7.939 | 14.795 | 21.220 | 22.752 | 26.225 |
| DNETD/10% | 5.043 | 6.151 | 7.363 | 8.149 | 14.320 | 21.168 | 20.946 | 24.797 |
| DNETD/20% | 5.280 | 7.388 | 7.631 | 7.734 | 15.028 | 25.682 | 20.143 | 25.215 |
| **NUMBER OF PIVOTS** | | | | | | | | |
| DNETD/MN | 198 | 343 | 183 | 247 | 477 | 719 | 513 | 574 |
| DNETD/FN | 682 | 323 | 1222 | 866 | 988 | 1374 | 1215 | 1190 |
| DNETD/10 | 404 | 296 | 395 | 461 | 751 | 940 | 924 | 848 |
| DNETD/40 | 232 | 288 | 250 | 322 | 573 | 682 | 696 | 632 |
| DNETD/100% | 175 | 351 | 193 | 236 | 492 | 743 | 537 | 574 |
| DNETD/5% | 516 | 309 | 757 | 461 | 671 | 782 | 823 | 745 |
| DNETD/10% | 404 | 317 | 395 | 428 | 573 | 682 | 696 | 632 |
| DNETD/20% | 315 | 343 | 291 | 322 | 504 | 721 | 563 | 546 |

TABLE VIII

CONTINUED

| PROBLEMS / STRATEGY | 4 | 9 | 11A | 12 | 14 | 14A | 14A.1 | 17 |
|---|---|---|---|---|---|---|---|---|
| **TOTAL NUMBER OF NODES IN SMALLER SUBTREE** | | | | | | | | |
| DNETD/MN | 2417 | 13467 | 2629 | 8184 | 27736 | 41641 | 31285 | 34787 |
| DNETD/FN | 3352 | 2973 | 4512 | 9280 | 17317 | 30948 | 23950 | 13787 |
| DNETD/10 | 2913 | 2991 | 3445 | 7077 | 19002 | 23006 | 23106 | 13808 |
| DNETD/40 | 3032 | 3176 | 3455 | 8338 | 11193 | 20469 | 19393 | 16913 |
| DNETD/100% | 1903 | 13722 | 2990 | 7266 | 30185 | 45813 | 31966 | 34787 |
| DNETD/5% | 3189 | 3881 | 4985 | 7077 | 13001 | 22196 | 22250 | 18643 |
| DNETD/10% | 2913 | 5075 | 3445 | 8060 | 11193 | 20469 | 19393 | 16913 |
| DNETD/20% | 3367 | 7151 | 3506 | 8338 | 12641 | 31215 | 15417 | 17023 |
| **TOTAL NUMBER OF BASIC ARCS WHOSE FLOWS WERE COMPUTED IN ORDER TO FIND THE LEAVING ARC** | | | | | | | | |
| DNETD/MN | 19701 | 102856 | 18216 | 49352 | 190722 | 287280 | 205086 | 229425 |
| DNETD/FN | 12184 | 9173 | 20890 | 43487 | 69150 | 98468 | 92529 | 67345 |
| DNETD/10 | 18003 | 19336 | 13134 | 32765 | 80508 | 116803 | 113829 | 85644 |
| DNETD/40 | 17383 | 36776 | 19525 | 37930 | 89609 | 136596 | 114655 | 108763 |
| DNETD/100% | 16990 | 104882 | 18139 | 46902 | 196498 | 279393 | 214385 | 228381 |
| DNETD/5% | 16360 | 28082 | 22759 | 32765 | 84096 | 123584 | 118865 | 103185 |
| DNETD/10% | 18003 | 41661 | 13134 | 41021 | 89609 | 136596 | 114655 | 108763 |
| DNETD/20% | 18922 | 59818 | 16250 | 37930 | 113938 | 190477 | 125284 | 134979 |

Looking at the solution times of Table VIII, no version clearly emerges as superior. On the 400 node problems, DNETD/40, which coincides with DNETD/10% for these problems, ran consistently faster than the other versions. On problems with a smaller number of nodes, DNETD/10% performed somewhat better than did DNETD/40. Overall, DNETD/10% solved problems more quickly than did the versions of DNETD employing the other percentage rules, losing only to DNETD/100% on problem 4 and to all of the other three on problem 12. In addition to outperforming DNETD/MN and DNETD/FN on the 400 node transshipment problems, DNETD/10% assumed a middle ground with respect to these two versions on the other problems. In short, the 10% rule emerges as a good consistent rule. For the remainder of the computational testing, the 10% rule was employed as the pivot selection criterion.

## 6.5 LIMITING THE LENGTH OF THE SORT LISTS

DNETD/10% requires 8 arc length and 7 node length arrays of storage. Since the three sort arrays are allocated $|E| - |N| + 1$ words of memory, we investigated the effect on total solution time of reducing the size of these arrays.

In solving the networks of Problem Set II, two other statistics (in addition to those previously mentioned) were compiled: the maximum and the average (per iteration) by each of the sort lists. TABLE IX contains these statistics obtained by using DNETD/10%:

TABLE IX

| PROBLEM | MAXIMUM SORT LISTS LENGTH | AVERAGE SORT LISTS LENGTH (PER ITERATION) |
|---|---|---|
| 4 | 111 | 7.4 |
| 9 | 23 | 2.0 |
| 11A | 117 | 15.4 |
| 12 | 50 | 6.3 |
| 14 | 64 | 4.2 |
| 14A | 54 | 3.6 |
| 14A.1 | 56 | 5.4 |
| 17 | 127 | 8.1 |

While the maximum sizes of these sort lists are rather large, the average sizes are quite small. In fact for these problems, the largest average sort list size was 15.4. We therefore restricted the maximum size of the lists to be a somewhat larger number, 25, and resolved the networks of Problem Set II. The solution times for the two versions of DNETD/10% are shown in TABLE X.

TABLE X

SOLVING PROBLEM SET II WITH DNETD/10%

| PROBLEMS | TOTAL SOLUTION TIME IN CP SECONDS | | NUMBER OF ITERATIONS | |
|---|---|---|---|---|
| | MAXIMUM SORT ARRAYS SIZE \|E\|-\|N\|+1 | 25 | MAXIMUM SORT ARRAYS SIZE \|E\|-\|N\|+1 | 25 |
| 4 | 5.043 | 5.826 | 404 | 462 |
| 9 | 6.151 | 5.922 | 317 | 317 |
| 11A | 7.363 | 7.751 | 395 | 522 |
| 12 | 8.149 | 7.546 | 428 | 399 |
| 14 | 14.320 | 18.112 | 573 | 733 |
| 14A | 21.168 | 19.175 | 682 | 642 |
| 14A.1 | 20.946 | 23.117 | 696 | 737 |
| 17 | 24.797 | 25.998 | 632 | 740 |

The results indicate that while a reduction in the size of the sort lists does not critically affect algorithmic performance, it drastically reduces storage requirements. In fact, on 3 of the 8 problems, the code employing the abbreviated sort lists actually solved the problems faster. This version of DNETD employing the abbreviated sort lists and the 10% pivot selection criterion will be called DNET2.

## 6.6 EVALUATION OF DNET2

To evaluate DNET2, we compared its performance on the 22 original problems with that of DNETD/FN, DNETD/MN, and the primal simplex transshipment code, PNET-I. [12, 14] (Since the problem networks

have many arcs out of the pure source nodes and few arcs into each node, the arc directions for every arc in each network were reversed during problem input for PNET-I so as to efficiently implement the pivot selection criterion (the most negative arc out of a node) used in PNET-I.) The results are shown in Table XI.

Note that DNET2 performed rather well in relation to DNETD/MN and DNETD/FN. Among these three versions, DNET2 ran fastest on eight problems and slowest on only one. On thirteen of the problems, its times fell between the times of the other two versions. These results indicate that, while DNET2 may not be the best dual code on a particular problem, its overall performance is quite consistent.

Comparing DNET2 to PNET-I, however, the primal code is uniformly superior although for 14 of the 22 problems, the primal code was less than 3 times faster and for 20 of the 22 problems, the primal code was less than 5 times faster.

In earlier testing in which the problem set consisted of sparse networks, PNET-I was consistently 6-12 times faster than the dual codes DNET and DNET-I [12, 15]. The sparsity of the networks was actually responsible for these dual codes performing as well as they did since both codes employ a full arc scan in order to determine the entering arc. The poor performance of DNETA in solving the current problem set's networks which feature a much higher arc to node ratio points out the computational expense of the full arc scan. Further, the more efficient functions and techniques presented in the previous sections indicate the superiority of DNET2 over the two earlier codes for the class of problems considered in this study.

TABLE XI

COMPARISONS OF DUAL AND PRIMAL CODES

CP TIME IN SECONDS ON CDC 6600

| | TOTAL SOLUTION TIME | | | | NUMBER OF PIVOTS | | | |
|---|---|---|---|---|---|---|---|---|
| PROBLEM | DNETD/MN | DNETD/FN | DNET2 | PNET-I | DNETD/MN | DNETD/FN | DNET2 | PNET-I |
| 1 | .808 | .568 | .648 | .141 | 91 | 91 | 88 | 105 |
| 2 | 1.023 | .637 | .662 | .260 | 105 | 110 | 94 | 176 |
| 3 | 2.343 | 2.517 | 2.248 | .798 | 128 | 360 | 208 | 487 |
| 4 | 5.219 | 4.986 | 5.826 | 2.234 | 203 | 682 | 462 | 988 |
| 5 | 9.106 | 7.610 | 5.154 | 4.493 | 291 | 999 | 329 | 1407 |
| 6 | 6.415 | 4.413 | 4.326 | 2.035 | 279 | 433 | 264 | 1084 |
| 7 | 10.432 | 6.703 | 6.565 | 3.518 | 298 | 476 | 291 | 1628 |
| 8 | 32.442 | 14.422 | 16.284 | 9.625 | 617 | 990 | 595 | 3438 |
| 9 | 8.604 | 5.122 | 5.934 | .662 | 328 | 322 | 317 | 269 |
| 10 | 52.604 | 36.486 | 28.315 | 9.709 | 783 | 1932 | 690 | 3622 |
| 11 | 5.139 | 6.205 | 5.874 | 3.162 | 184 | 847 | 408 | 2323 |
| 11A | 5.390 | 8.531 | 7.748 | 4.094 | 188 | 1222 | 522 | 2584 |
| 12 | 7.131 | 10.621 | 7.553 | 2.931 | 244 | 886 | 399 | 1787 |
| 12A | 7.932 | 12.767 | 9.685 | 3.677 | 259 | 894 | 474 | 2217 |
| 13 | 18.008 | 23.943 | 19.992 | 3.708 | 481 | 1435 | 771 | 2095 |
| 13A | 20.906 | 33.955 | 26.668 | 6.344 | 468 | 1946 | 1053 | 2903 |
| 14 | 20.010 | 15.957 | 18.249 | 3.956 | 500 | 988 | 733 | 2254 |
| 14A | 27.017 | 22.590 | 19.213 | 6.116 | 707 | 1215 | 642 | 3047 |
| 15 | 21.301 | 15.848 | 14.769 | 3.721 | 500 | 932 | 535 | 2092 |
| 16 | 20.204 | 15.419 | 15.098 | 3.469 | 521 | 894 | 614 | 2019 |
| 16A | 28.850 | 20.412 | 20.132 | 7.845 | 619 | 1166 | 645 | 3815 |
| 17 | 34.813 | 25.471 | 26.051 | 9.942 | 556 | 1190 | 740 | 4199 |

In order to measure the effect of problem variation on code performance, we then compared the performance of DNET2 to that of PNET-I when the following problem parameters were varied: total supply, upper bound values, cost ranges, and the number of arcs in the network.

When the original problem set was constructed, the supply, and upper bound values were set so that the flow on many nonbasic arcs would be at their upper bounds at optimality. The results in Table XII indicate that this indeed occurred; for the 22 problems the percentage of nonbasic arcs at their upper bounds ranged from 28% to 59%.

We first considered the effect of reducing the total supply and constructed five new transshipment problems. The specifications of these problems are identical to those of problems 11, 12, 13, 14, and 14A except that total supply is reduced by a factor of 10. The computational results appear in Table XIII.

For the modified problems the percentage of nonbasic arcs having flow values equal to their upper bounds diminished by 300% to 400%. While the solution times for DNET2 tended to decrease, those for PNET-I decreased somewhat more.

We felt that simply increasing arc upper bound values by an order of magnitude from 5 - 10 to 50 - 100 would have the same effect as reducing total supply. We thus decided to test the effect of decreasing total supply while also increasing upper bound values. Accordingly, the modified problems used in the total supply testing were further changed so that the upper bound values now ranged from 50 - 100. Table XIII contains the results of this testing.

TABLE XII

CHARACTERISTICS OF SOLUTIONS
OBTAINED WITH DNET2

CP TIME IN SECONDS ON CDC 6600

| PROBLEM | TOTAL SOLUTION TIME | START TIME | START TIME AS PERCENTAGE OF TOTAL SOLUTION TIME | NUMBER OF NONBASIC ARCS AT UPPER BOUND AT OPTIMALITY | PERCENTAGE OF NONBASIC ARCS AT UPPER BOUND AT OPTIMALITY |
|---|---|---|---|---|---|
| 1 | .648 | .353 | .54 | 54 | .31 |
| 2 | .662 | .403 | .61 | 131 | .55 |
| 3 | 2.248 | .873 | .39 | 314 | .40 |
| 4 | 5.826 | 1.649 | .28 | 595 | .34 |
| 5 | 5.154 | 2.131 | .41 | 821 | .35 |
| 6 | 4.326 | 2.421 | .56 | 438 | .42 |
| 7 | 6.565 | 3.921 | .60 | 785 | .39 |
| 8 | 16.284 | 8.199 | .50 | 1365 | .28 |
| 9 | 5.934 | 3.582 | .60 | 486 | .59 |
| 10 | 28.315 | 16.845 | .59 | 1617 | .35 |
| 11 | 5.874 | 1.477 | .25 | 654 | .44 |
| 11A | 7.748 | 1.778 | .23 | 682 | .36 |
| 12 | 7.553 | 2.991 | .40 | 545 | .39 |
| 12A | 9.685 | 3.673 | .38 | 805 | .45 |
| 13 | 19.992 | 7.979 | .40 | 561 | .33 |
| 13A | 26.668 | 9.728 | .36 | 844 | .39 |
| 14 | 18.249 | 8.328 | .46 | 598 | .35 |
| 14A | 19.213 | 11.144 | .58 | 1019 | .38 |
| 15 | 14.769 | 8.237 | .56 | 558 | .33 |
| 16 | 15.098 | 8.126 | .54 | 749 | .45 |
| 16A | 20.132 | 10.946 | .54 | 1141 | .44 |
| 17 | 26.051 | 13.827 | .53 | 1385 | .38 |

TABLE XIII

SENSITIVITY ANALYSIS OF PRIMAL AND DUAL CODES
CP TIME IN SECONDS ON CDC 6600

VARYING TOTAL SUPPLY

| PROBLEMS: | 11 | 11B | 12 | 12B | 13 | 13B | 14 | 14B | 14A | 14A.1 |
|---|---|---|---|---|---|---|---|---|---|---|
| TOTAL SOLUTION TIME | | | | | | | | | | |
| DNET2 | 5.874 | 4.851 | 7.553 | 7.691 | 19.992 | 18.648 | 18.249 | 15.778 | 19.213 | 23.117 |
| PNET-I | 3.162 | 1.672 | 2.931 | 2.132 | 3.708 | 5.118 | 3.956 | 3.602 | 6.116 | 5.646 |
| NUMBER OF PIVOTS | | | | | | | | | | |
| DNET2 | 408 | 453 | 399 | 524 | 771 | 716 | 733 | 578 | 642 | 737 |
| PNET-I | 2323 | 1036 | 1787 | 1225 | 2095 | 1999 | 2254 | 1689 | 3047 | 2029 |

VARYING UPPER BOUNDS

| PROBLEMS: | 11B | 11C | 12B | 12C | 13B | 13C | 14B | 14C | 14A.1 | 14A.2 |
|---|---|---|---|---|---|---|---|---|---|---|
| TOTAL SOLUTION TIME | | | | | | | | | | |
| DNET2 | 4.851 | 2.423 | 7.691 | 6.893 | 18.648 | 20.817 | 15.778 | 18.235 | 23.117 | 24.110 |
| PNET-I | 1.672 | .611 | 2.132 | 1.474 | 5.118 | 2.800 | 3.602 | 3.770 | 5.646 | 3.396 |
| NUMBER OF PIVOTS | | | | | | | | | | |
| DNET2 | 453 | 110 | 524 | 275 | 716 | 573 | 578 | 477 | 737 | 515 |
| PNET-I | 1036 | 315 | 1225 | 766 | 1999 | 1177 | 1689 | 1650 | 2029 | 1209 |

VARYING COST RANGES

| PROBLEMS: | 4 | 4E | 9 | 9E | 11 | 11E | 13 | 13E |
|---|---|---|---|---|---|---|---|---|
| TOTAL SOLUTION TIME | | | | | | | | |
| DNET2 | 5.826 | 4.328 | 5.934 | 6.097 | 5.874 | 5.688 | 19.992 | 20.155 |
| PNET-I | 2.234 | 1.976 | .662 | 1.029 | 3.162 | 2.775 | 3.708 | 4.357 |
| NUMBER OF PIVOTS | | | | | | | | |
| DNET2 | 462 | 333 | 317 | 324 | 408 | 361 | 771 | 741 |
| PNET-I | 988 | 829 | 269 | 431 | 2323 | 2035 | 2095 | 2067 |

VARYING ARC DENSITY

| PROBLEMS: | 11 | 11A | 12 | 12A | 13 | 13A | 14 | 14A | 17 |
|---|---|---|---|---|---|---|---|---|---|
| TOTAL SOLUTION TIME | | | | | | | | | |
| DNET2 | 5.917 | 7.751 | 7.546 | 9.685 | 19.957 | 26.411 | 18.112 | 19.175 | 25.998 |
| PNET-I | 3.162 | 4.094 | 2.931 | 3.677 | 3.708 | 6.344 | 3.956 | 6.116 | 9.942 |
| NUMBER OF PIVOTS | | | | | | | | | |
| DNET2 | 408 | 411 | 399 | 474 | 771 | 1053 | 733 | 642 | 740 |
| PNET-I | 2323 | 2584 | 1787 | 2217 | 2095 | 2903 | 2254 | 3047 | 4199 |

The optimal solutions to these networks have very few nonbasic arcs at their upper bounds, approximately 1%. Concurrent with this drop, the maximum sort array size dropped for four of the five networks to under 25. Hence, no large multiple pivots were performed for these networks. For this testing the times for DNET2 tended to increase slightly while those for PNET-I decreased.

The net effect of increasing the upper bound values and decreasing the total supply upon the dual times was nearly negligible. However, the primal times dropped by nearly 100% so that on these less restricted networks the primal was approximately 6 times faster than the dual. For the five problems of the original 22 problems, the primal was less than four times faster.

The effect of varying cost ranges was considered next. For four of the original problems, the arc cost ranges were increased from 1 - 100 to 1 - 1000. The results shown in Table XVII were inclusive for both PNET-I and DNET2. Both codes performed better on two problems with the larger cost range and worse on the other two problems. In general, however, the degree of difference was slight for both codes, indicating that both codes are relatively insensitive to cost ranges.

The sensitivity of the two codes to the number of arcs in a network was then considered. Of the original 22 problems, 5 have, with the exception of the number of arcs desired, the same specifications of some other problem in the set. Problems 11A and 12A have the same specifications as 11 and 12 except that the latter have 1600 arcs while the former have 2000 arcs. Problem 13 has 2000 arcs while 13A has 2500.

Problem 14 has 2000 arcs, 14A has 3000, and 17 has 4000 arcs. The solution statistics for these problems are presented in Table XIII.

The times for both PNET-I and DNET2 increase with arc density. However, PNET-I was more sensitive to large increases in the number of arcs. The times for DNET2 increased by only 44% when the number of arcs were doubled from Problem 14 to Problem 17, while the times for PNET-I increased by 150%. This sensitivity of the primal is due to the pivot selection rule employed and the fact that PNET-I is designed for sparse networks.

## 7.0 CONCLUDING REMARKS

In this paper we have presented an efficient implementation of the modified dual simplex algorithm for capacitated transshipment problems. Compared to prior dual simplex transshipment codes, this new implementation features improved list structures to represent the basis and to store the original problem data. These list structures access desired data so that the processing of superfluous data is minimized. Further, the dual simplex method was modified to combine, under certain conditions, several iterations into one. We formulated and tested three implementations, each of which sotres the original problem data somewhat differently, and found one approach to be superior. Using this approach, we then proceeded to evaluate various pivot selection criteria and test modifications of the multiple pivot strategy. The resultant code, which we deemed to be the "best" implementation of the dual method, is called DNET2.

The literature has indicated the superiority, both in terms of

storage and time, of special purpose primal based simplex approaches over dual and out-of-kilter methods [2, 3, 12]. Our testing, which compared the performance of DNET2 to that of a state-of-the-art primal code, PNET-I, has not suggested otherwise. However, we were able to isolate a problem topology for which the times of DNET2 only are 2 to 3 times slower than those of PNET-I. This topology features very dense networks having few sources, many sinks, large total supply requirements, and small arc flow ranges; for such a topology earlier testing [2] indicated the unsuitability of SUPERK, a state-of-the-art out-of-kilter code. Consequently we did not use SUPERK in the comparative study.

Analyzing the performance of DNET2 further, as Table XII indicates, a considerable percentage of the total solution time is spent in finding an initial dual feasible solution. While further research may be undertaken to find improved starting techniques for the dual method, it does not appear possible to make the dual superior to the primal for solving transshipment problems which do not have a good dual feasible solution readily available. Often branch-and-bound or post-optimality procedures require the altering of parameter values and then the re-solving of the model. The previous optimal basis may still yield a dual feasible solution with respect to the new parameters. For these situations, this dual code may be particularly effective.

## REFERENCES


1. Balas, E.and Zemel, E., "Solving Large Zero-One Knapsack Problems," Management Sciences Report No. 408, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, June 1976, Reused July 1977.

2. Barr, R. S., Elam, J., Glover, F., and Klingman, D., "A Network Augmenting Path Basis Algorithm for Transshipment Problems."

3. Barr, R. S., Glover, F., and Klingman, D., "An Improved Version of the Out-of-Kilter Method and a Comparative Study of Computer Codes," Mathematical Programming 7, (1974), 60-86.

4. Barrodale, I. and Roberts, F. D. K., "An Improved Algorithm for Discrete $L_1$ Linear Approximation," SIAM Journal of Numerical Analysis, 10 (October, 1973), 839-848

5. Charnes, A. and Cooper, W. W., Management Models and Industrial Applications of Linear Programming, 2 vols., Wiley, New York, 1961.

6. Dantzig, G. Linear Programming and Extensions, Princeton University Press, Princeton, N.J., 1963.

7. Dial, R., Glover, F., Karney, D., and Klingman, D., "A Computational Analysis of Alternative Algorithms and Labeling Techniques for finding Shortest Path Trees," Research Report CCS291, Center for Cybernetic Studies, The University of Texas at Austin, 1977. To appear in Networks.

8. Gilsinn, J. and Witzgall, C., "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," NBS Technical Note 772, U.S. Department of Commerce, 1973.

9. Glover, F., Hultz, J., Klingman, D., "Generalized Networks: A Fundamental Computer-Based Planning Tool," Management Science, 24, 12 (1978), 1209-1220.

10. Glover, F., Karney, D., and Klingman, D. "The Augmented Predecessor Index Method for Locating Stepping Stones Paths and Assigning Dual Prices in Distribution Problems," Transportation Science, 6(1), 1972, pp 171-180.

11. Glover, F., Karney, D., and Klingman, D., "Double-Pricing Dual and Feasible Start Algorithms for the Capacitated Transportation (Distribution) Problem," Research Report CCS105, Center for Cybernetic Studies, University of Texas at Austin, 1973.

12. Glover, F., Karney, D., and Klingman, D., "Implementation and Computational Comparisons of Primal, Dual, and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," Networks 4, 3(1974), 191-212.

13. Glover, F., Karney, D., and Klingman, D., and Napier, A., "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Management Science 20,5 (1974), 793-813.

14. Glover, F., Klingman, D., and Stutz, J., "Augmented Threaded Index Method for Network Optimization," INFOR 12, 3 (1974), 293-298.

15. Hultz, J., Klingman, D., and Russell, R., "An Advanced Dual Basic Feasible Solution For a Class of Capacitated Generalized Networks," Operations Research 24, 2, (March-April, 1976), 301-313.

16. Jacobsen, S. K., "On the Use of Tree Indexing Methods in Transportation Algorithms," European Journal of Operational Research 2, 1(Jan., 1978), 54-65.

17. Johnson, E., "Networks and Basic Solutions," Operations Research 14(1966), 619-623.

18. Klingman, D., Napier, A., and Stutz, J., "NETGEN- A Program for Generating Large Scale (Un) Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems," Management Science 20,5 (Jan., 1974), 814-821.

19. Wagner, H. "The Dual Simplex Algorithm for Bounded Variables," Naval Research Logistics Quarterly 5, 3, (1958), 257-261.

20. Witzgall, C., "On One-Row Linear Programs," International Symposium or Extremal Methods and Systems Analysis on the Occasion of Professor A. Charnes' 60th Birthday, University of Texas at Austin, 1977.

Unclassified

Security Classification

**DOCUMENT CONTROL DATA - R & D**

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

| 1. ORIGINATING ACTIVITY (Corporate author) | 2a. REPORT SECURITY CLASSIFICATION |
|---|---|
| Center for Cybernetic Studies<br>The University of Texas at Austin | Unclassified |
| | 2b. GROUP |

3. REPORT TITLE

Implementation and Analysis of a Variant of the Dual Method for the Capacitated Transshipment Problem

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

Ronald D. Armstrong
Darwin D. Klingman
David H. Whitman

| 6. REPORT DATE | 7a. TOTAL NO. OF PAGES | 7b. NO. OF REFS |
|---|---|---|
| October 1978 | 61 | 20 |

| 8a. CONTRACT OR GRANT NO. | 9a. ORIGINATOR'S REPORT NUMBER(S) |
|---|---|
| N0014-78-C-0222; NSF MCS77-00100 | Center for Cybernetic Studies<br>Research Report CCS 324 |
| b. PROJECT NO.<br>NR047172 | |
| c. | 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) |
| d. | |

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited

| 11. SUPPLEMENTARY NOTES | 12. SPONSORING MILITARY ACTIVITY |
|---|---|
| | Office of Naval Research (Code 434)<br>Washington, D.C. |

13. ABSTRACT

This paper presents a variant of the dual simplex method for the capacitated pure network problem and a computational analysis of this algorithm. This work includes the considerations of different list structures to store the original problem data and the basis and the testing of various procedures to select the leaving basic variable. This study also examines the sensitivity of the implementation to changes in problem parameters. The results show that the algorithm which is presented here is superior to earlier dual implementations.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

Unclassified

Security Classification

A-31408

| 14 KEY WORDS | LINK A | | LINK B | | LINK C | |
|---|---|---|---|---|---|---|
| | ROLE | WT | ROLE | WT | ROLE | WT |
| Networks | | | | | | |
| Dual Method | | | | | | |
| Linked Lists | | | | | | |
| Transshipment Problems | | | | | | |
| Linear Programming | | | | | | |